REDUCING THE ENCODING TIME OF MOTION ESTIMATION IN HEVC USING PARALLEL PROGRAMMING

by

VASAVEE VIJAYARAGHAVAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2015

Acknowledgements

First and foremost, I would like to thank Dr. K. R. Rao for guiding and mentoring me and for being a constant source of encouragement throughout my thesis. He has been a pillar of support throughout my Masters degree program and a great guide for my thesis research. I would like to thank Dr. Jonathan Bredow and Dr. Ioannis D. Schizas for serving on my committee.

My sincere thanks to Kreig Dubose, Jim R Blakley and Mike Downs from Intel Corporation who have been supportive technically and emotionally to say “Yes you will successfully complete your thesis with flying colors” and huge thanks to Sravanthi Kota Venkata for the immense knowledge and help delivered by her that aided towards successful completion of my thesis.

I would like to thank my friend Karthik Arunachalam for providing me with strong fundamentals on parallel programming and my MPL lab mates for providing valuable inputs throughout my research.

Last but never the least: I would like to thank my family and friends for being the biggest support during the course of my Masters and my thesis research and without whom I would not have been here today.

November 24, 2015

Abstract

REDUCING THE ENCODING TIME OF MOTION ESTIMATION IN HEVC USING PARALLEL PROGRAMMING

VASAVEE VIJAYARAGHAVAN, MS

The University of Texas at Arlington, 2015

Supervising Professor: K.R.Rao

High Efficiency Video Coding (HEVC) [10] is the current state-of-art video codec which is widely being adopted by lot of users. It has close to 50% reduction in encoding time compared to its predecessor, H.264 or AVC [37] (Advanced Video Coding) at the cost of increased complexity. Lot of research is going towards reducing the complexity of this codec, at the same time, maintaining the visual quality that it produces and maintaining the reduced encoding time from its predecessor.

As an effort to decrease the encoding time further, there can be several approaches. Parallel processing is taking a dominant role in many places, especially in Graphics Processing Unit (GPU) and multi-cored processor based applications. Because of the ability of the parallel programming to utilize the multiple cores efficiently at the same time, in place of serial programming, this has been used in many applications which demand quicker completion.

If areas that are parallelizable are identified in any codec [38] (HEVC in this case), the encoding time can be drastically reduced by writing an efficient algorithm.

In parallel programming, it is very important that the parallelized portion has the least amount of dependencies; otherwise it will lead to reverse effects of what is actually expected.

Thus, the success lies in identifying the region of the codec that contributes more towards encoding time and that has least dependencies, and optimizing that portion of the codec. In this thesis, thorough analysis is done to identify the hot spots in the codec implementation, HM16.7, of High Efficiency Video Coding (HEVC) developed by the JCTVC team. This hotspot analysis is implemented using Intel's most powerful tool, Intel® vTune™ Amplifier. The results of this hotspot analysis will be functions and loops that use most of the CPU time. Once this is identified, the respective function is targeted to be optimized using Parallel programming with OpenMP. Iterative runs are carried out on the modified code to check whether the code has been reasonably optimized. The final optimized code is tested for encoding videos using metrics such as PSNR (Peak Signal to Noise Ratio), R-D plot (Rate Distortion) and computational complexity in terms of encoding time.

Through optimization of the HEVC HM16.7 encoder, there is an average reduction of ~24.7% to ~42.3% in encoding time with ~3.5 to 7% PSNR gain and ~1.6% to 4% bitrate increase.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

INTRODUCTION

1.1 Motivation

In today's technological world, the demand for videos is increasing at a dramatic rate, as the number of electronic devices become more and as they become very easy to use. At the same time, bandwidth requirements are never a factor that would go down. It rather keeps exploding as the need for videos to be watched over the web keeps increasing. There has been development of different video codecs by different companies, each of them trying to optimize the codec over the previous version. The better the coding algorithm, lesser might be the requirement for bandwidth to transmit the video. This again depends on multiple factors. This efficiency of the codec should not come at the cost of video quality. Some factors that are taken into consideration while designing a video codec are:

- Encoding Time.
- Video Quality (Measured by using objective measurement metrics such as PSNR, SSIM, BDRATE etc).
- File size of the encoded video (More the file size, better will be the video quality.)

These factors directly influence:

- Bandwidth requirement over the network.
- Quality of video watched by the user.
- Storage capacity of any server that stores and transmits the encoded video.
- Storage capacity of device that records and stores the compressed video.

High Efficiency Video Coding (HEVC) [10] is the current state-of-art video codec which is widely being adopted by lot of users. It has close to 50% reduction in encoding time compared to its predecessor, H.264 or AVC [37] (Advanced Video Coding) at the cost of increased complexity. Lot of research is going towards reducing the complexity of this codec, at the same time, maintaining the visual quality that it produces and maintaining the reduced encoding time from its predecessor.

1.2 Background Work

As an effort to decrease the encoding time further, there can be several approaches. Parallel processing is taking a dominant role in many places, especially in Graphics Processing Unit (GPU) and multi-cored processor based applications. Because of the ability of the parallel programming to utilize the multiple cores efficiently at the same time, in place of serial programming, this has been used in many applications which demand quicker completion.

If areas that are parallelizable are identified in any codec [38] (HEVC in this case), the encoding time can be drastically reduced by writing an efficient algorithm.

In parallel programming, it is very important that the parallelized portion has the least amount of dependencies, otherwise it will lead to reverse effects of what is actually expected.

Thus, the success lies in identifying the region of the codec that contributes more towards encoding time and that has least dependencies, and optimizing that portion of the codec.

1.3 Thesis Outline

In this thesis, efforts have been made to identify the hotspots in the HEVC [10] code and the tools that have been used for this will be explained in detail in the chapters that follow. Also, studies have been made to identify the region of the code (functions) which are most parallelizable with least dependencies. Hence, the function which is to be optimized is identified (Figure 1.3.1). Optimization is achieved by using parallel programming on CPU + GPU based systems, keeping the serial code running in the CPU while launching the parallel code on the GPU.

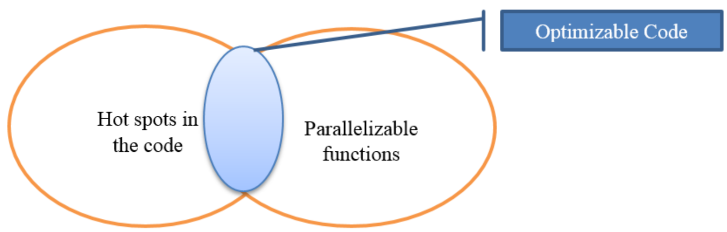


Figure 1-1 Identifying the region to be optimized in any given codec

1.4 Organization of this thesis

The following chapters of the report is organized in the following manner:

The need for video coding and an introduction to the same is explained in CHAPTER 2, followed by a brief introduction to High Efficiency Video Coding in CHAPTER 3. Detailed explanation of how to identify the region of the code to be optimized is explained in CHAPTER 4. An introduction to motion estimation in HEVC is given in CHAPTER 5 followed by an introduction to Parallel Programming in CHAPTER 6.The rest of the CHAPTERs from 7 to 10 explain the algorithm adopted in this thesis, experimental conditions, results, metrics used for comparison of obtained results and future work ending with references.

Chapter 2

GROWING NEED FOR VIDEO CODECS

2.1 Where do we use videos?

Almost ubiquitous everywhere!!!

We record videos and photos in our mobile phones. Try to upload them in YouTube or Facebook or send them through Skype or Whatapp! Something which we do on a day to day basis. We never realize how much of Internet traffic this uploading and downloading of videos/images consume. This is just us, the consumers.

Providers take the top seat in consuming the internet traffic. Broadcasters have challenges henceforth, in delivering quality videos to all of their customers.

The number of mobile devices have exploded. Personal computers (PCs) have become less existent and laptops and tablets have become the most convenient devices to carry wherever we go.

The challenge lies in matching the network traffic and bandwidth requirements on par with the growing number of portable electronic devices. Let us take a look at Internet traffic – something that is most spoken among the media folks in the industry.

2.2 Top Providers that consume the most of Internet Traffic [2]

Figure 2-1 Top consumers of Internet traffic [2]

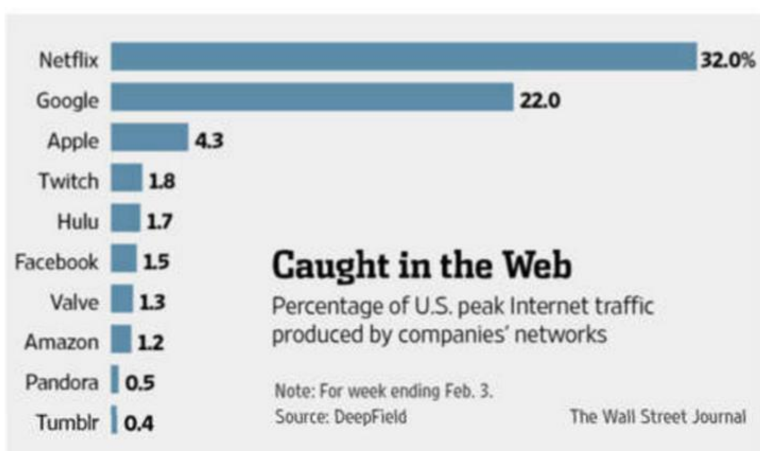


Figure 2-2 Top Internet Traffic produced by Corporations in 2014 [2]

2.3 Bandwidth Explosion – The Hottest Topic in Media Today [1] - [5]

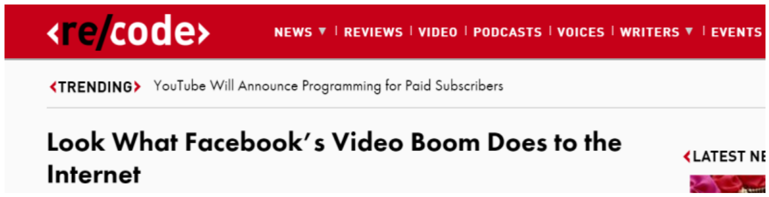


Figure 2-3 Facebook's video boom [1] – [5]

Figure 2-4 Bandwidth Explosion [1] – [5]

Figure 2-5 Mobile bandwidth requirements driven up by OTT streaming [1] - [5]

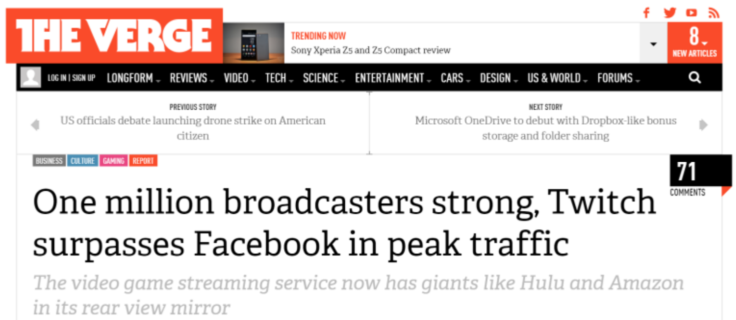


Figure 2-6 Twitch contributing to Internet traffic [1] – [5]

Figure 2-7 Netflix being the source of internet traffic [1] – [5]

The amount of videos watched by users in different resolutions through different electronic devices is exploding every year. Studies are being conducted by several organizations, which focus on network traffic and bandwidth consumption.

Here is a chart from Sandvine, the broadband network company [1]:

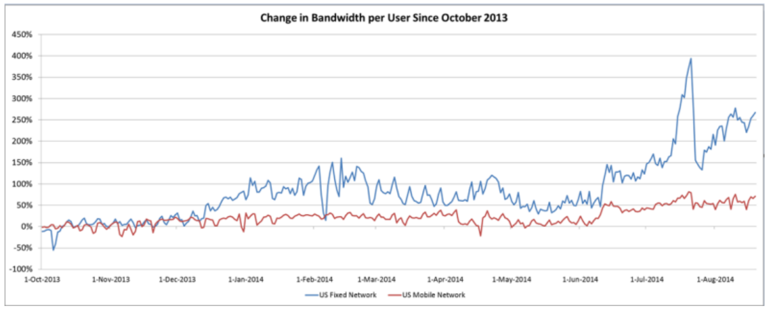


Figure 2-8 Change in Bandwidth per User since October 2013 by Sandvine [1]

How worse will this scenario get, if users/providers start using raw videos? Let us see some numbers on comparison between raw video file size and compressed video file size.

2.4 That is why we need Video Compression!!

Consider a digital video sequence having a picture resolution of 720x480 and a frame rate of 30 frames per second (FPS). If a picture is represented using the YUV color space with 8 bits per component or 3 bytes per pixel, size of each frame is 720x480x3 bytes. The disk space required to store one second of video is 720x480x3x30 = 31.1 MB. A one hour video would thus require 112 GB.

With the number of devices inside household increasing, the bandwidth requirement is also increasing. In addition to these extremely high storage and bandwidth requirements, using uncompressed video will add significant cost to the hardware and systems that process digital video.

Digital video compression with the help of video codecs is thus necessary even with exponentially increasing bandwidth and storage capacities. Fortunately, digital video has significant redundancies and eliminating or reducing those redundancies results in compression.

Video compression is typically achieved by exploiting

1. Spatial
2. Temporal
3. Statistical and psycho-visual redundancies

2.5 Introduction and Evolution of Video Coding Standards [6]

Every video coding standards adopt compression strategy to compress every video.

Table 2-1 Compression Strategies [19]

Information Type	Compression Tool
Spatial Redundancy	Intra prediction
Perceptual Redundancy	HVS based Quantization
Statistical Redundancy	Entropy Coding
Temporal Redundancy	Inter prediction

2.5.1 Spatial Redundancy Removal

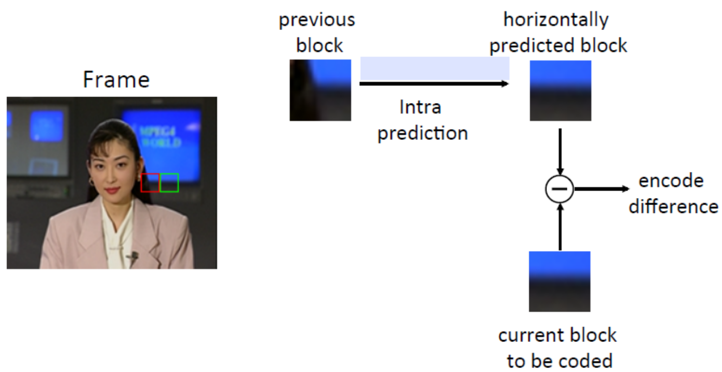


Figure 2-9 Spatial Redundancy Removal using Intra Prediction [19]

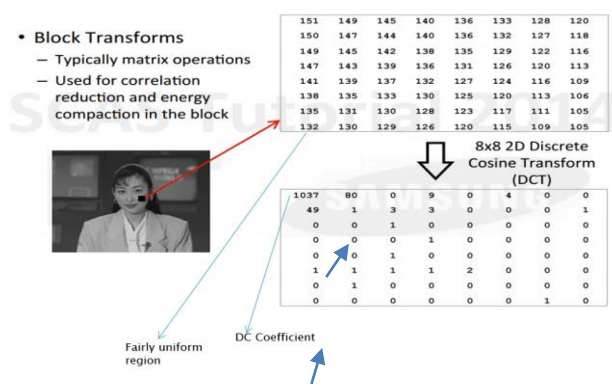


Figure 2-10 Spatial Redundancy Removal using block transforms [19]

2.5.2 Perceptual Redundancy Removal [19]

Human visual system is more sensitive to low frequency information. Perceptual redundancy removal makes use of this. Not all video data are equally significant from a perceptual point of view.

Figure 2-11 HVS more sensitive to low frequencies – Perceptual Redundancy

[19]

Quantization is a good tool for perceptual redundancy removal. Most significant bits (MSBs) are perceptually more important than least significant bits (LSBs). Co-efficient dropping (quantization with zero bits) example is shown in Figure 2-12:

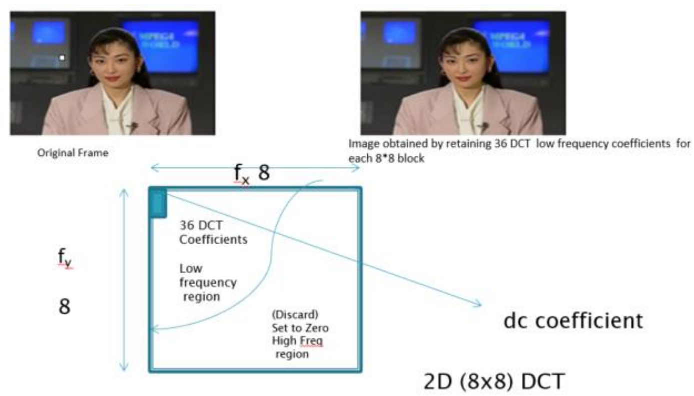


Figure 2-12 Quantization with zero bits [19]

2.5.3 Statistical Redundancy Removal [19]

Not all pixel values in an image (or in the transformed image) occur with equal probability. Entropy coding (eg. Variable length coding) can be used to represent more frequent values using shorter codewords and less frequently used values with longer codewords.

Different entropy coding includes:

Huffman coding

Golomb code

Arithmetic code

Rice code

Tunstall code

$$1^{st} \text{ order Entropy} = -\sum_{N} P_i \log_2 P_i$$

Pi is the probability of occurrence of symbol i, i= 1,2,3,…,N

Minimum theoretical bit rate at which a group of N symbols can be coded.

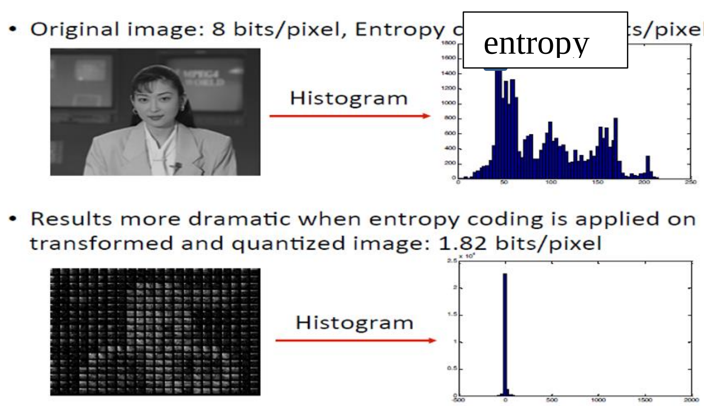


Figure 2-13 Statistical redundancy removal using entropy coding technique [19]

2.5.4 Temporal Redundancy Removal [19], [20]

Inter prediction is used in temporal redundancy removal. Frame difference can be coded using DCT and then can be quantized and entropy encoded.

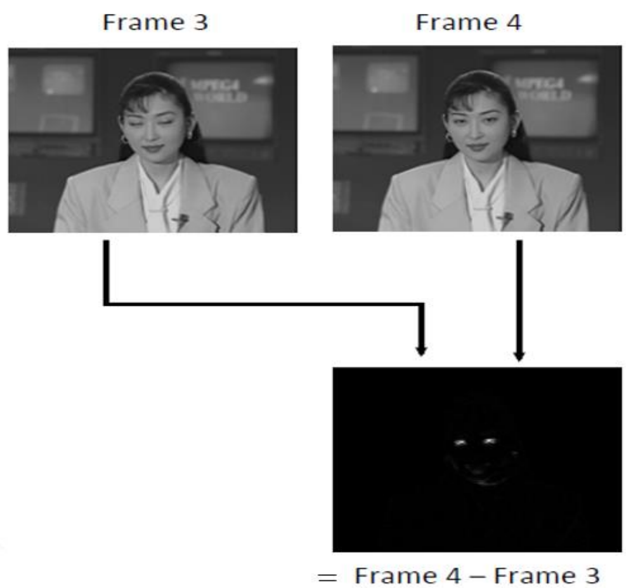


Figure 2-14 Frame difference used for temporal redundancy removal [19]

Inter prediction is implemented using motion compensation. Each frame of a video is divided into blocks and motion estimation/compensation is applied. For each block, the relative motion between the c Frame difference ing block of the same size in the previous frame is found out. Motion vectors are transmitted for each block. This is shown in Figure 2-9:

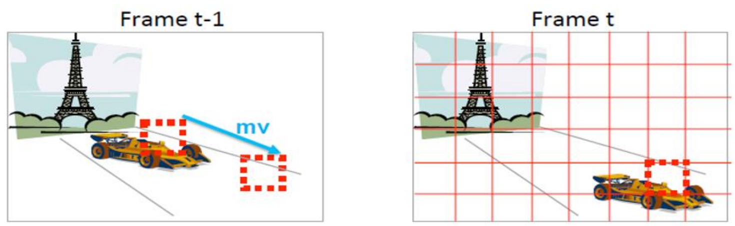


Figure 2-15 Motion compensated prediction [19], [20]

2.6 Temporal Prediction and Picture Coding Types [19]

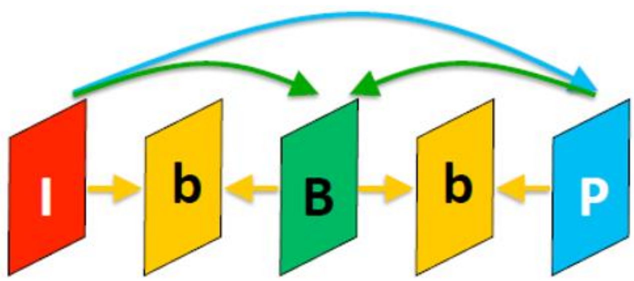


Figure 2-16 Picture Coding Types [19]

Intra Picture (I) – Picture is coded without reference to other pictures.

Inter Picture (P, B, b):

Uni-directionally predicted (P) Picture – Picture is predicted from one prior coded picture

Bi-directionally predicted (B, b) Picture – Picture is coded from one prior coded and one future coded pictures (b picture is not used as reference).

2.7 Summary of Key steps in video coding

Step 1: Intra and Inter prediction

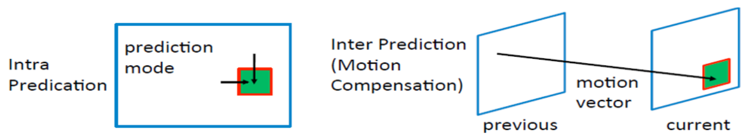


Figure 2-17 Intra and inter prediction modes [19]

Step 2: Transform and Quantization of residual (prediction error)

Figure 2-18 Transform and Quantization [19]

*Residual Figure from J.Apostolopoulos, " video Compression," MIT 6.344 Lecture, Spring 2004.

Step 3: Entropy coding on syntax elements (e.g.prediction modes, motion vectors, coefficients)

Step 4: In-loop filtering to reduce coding artifacts

2.8 Video Compression Standards [19]

Video compression standards ensure inter-operability between encoder and decoder. They usually support multiple use cases and applications by introducing different levels and profiles. Video coding standards specifies decoder mapping of bits to pixels. There has been close to ~2x improvement in compression from one standard to the next every decade.

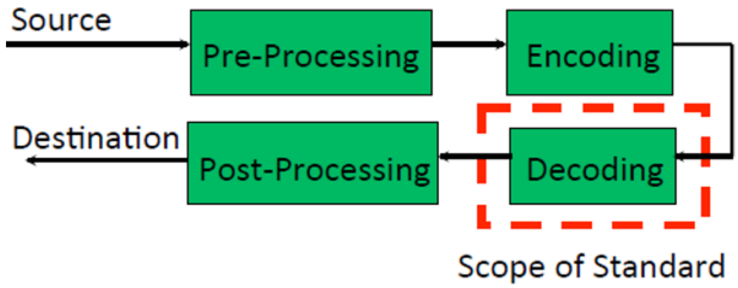


Figure 2-19 Video processing loop [19]

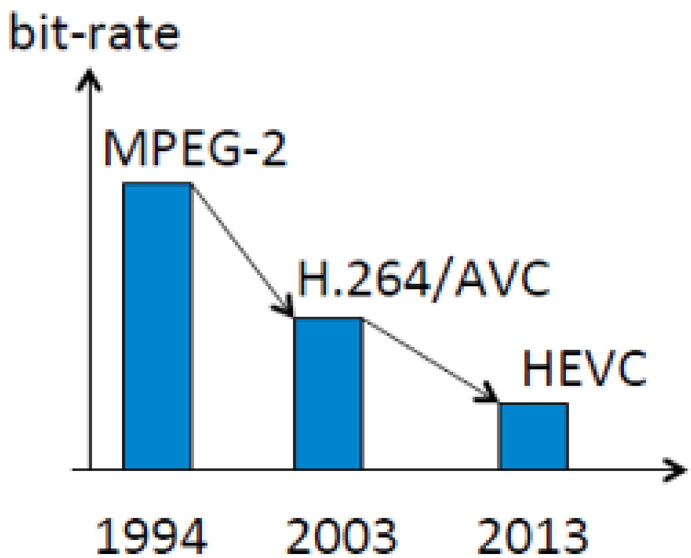


Figure 2-20 Bitrate reduction achieved for every new Video Coding Standard [19]

2.9 History of Video Coding Standards [19]

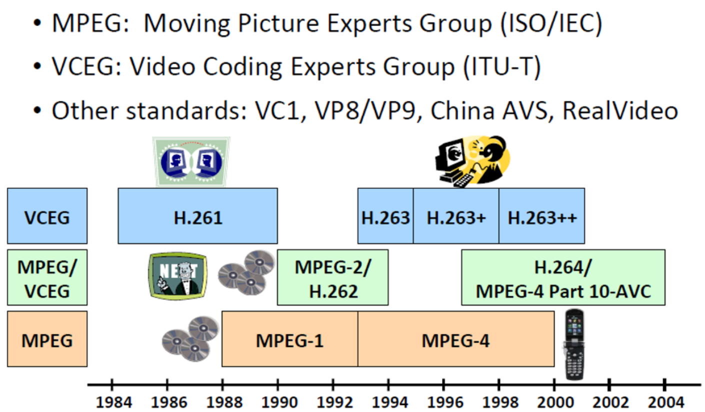


Figure 2-21 History of Video Coding Standards [19]

2.10 Evolution of Video Coding Standards [7], [19]

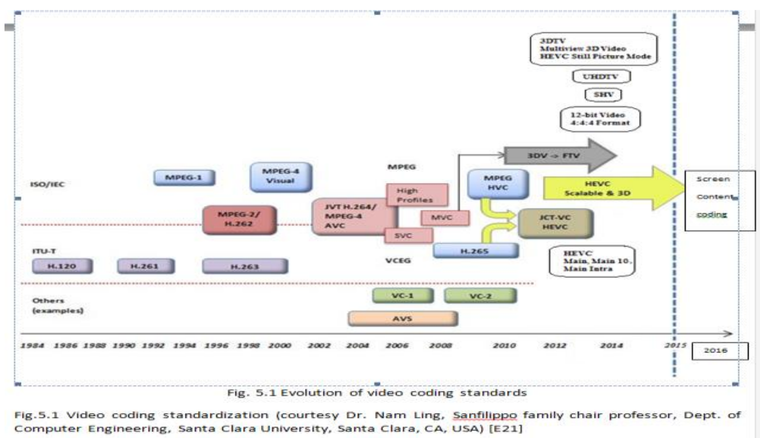


Figure 2-22 Video coding standardization upto early 2015 [19]

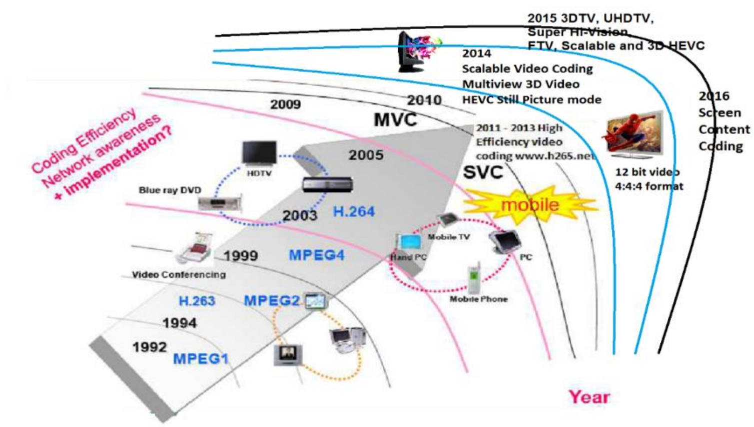
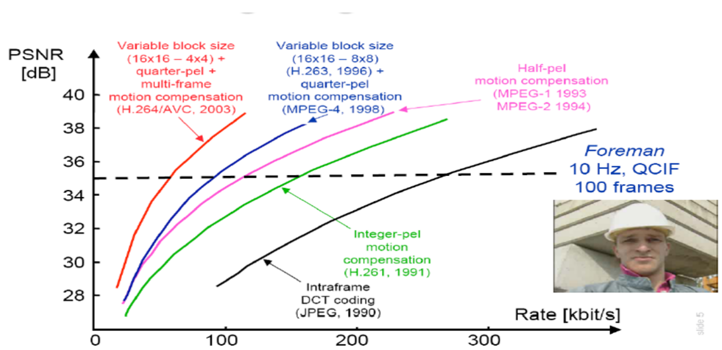


Figure 2-23 Evolution of Video Coding Standards [7]



QCIF : Quarter Common Intermediate Format

Figure 2-24 Progress in Video Coding [19]

2.11 Video Coding Standards and Applications [19]

Table 2-2 Different Video Coding Standards and Applications

STANDARD	MAIN APPLICATIONS	YEAR
JPEG,JPEG2000	Image	1992–1999 (JPEG), 2000 (JPEG2000)
JBIG	Fax	1995–2000
H.261	Video conferencing	1990
H.262, H.262+	DTV, SDTV	1995, 2000
H.263, H.263+	Videophone	1998,2000
MPEG-1	Video CD	1992
MPEG-2	DTV, SDTV, HDTV, DVD	1995
MPEG-4	Interactive video	2000
MPEG-7	Multimedia content description interface	2001
MPEG-21	Multimedia framework	2002
H.264/ MPEG-4 Part 10	Advance video coding	2003
	Fidelity range extensions (high profile), Studio editing, Post processing, Digital cinema	August, 2004

JPEG,JPEG2000	Image	1992–1999 (JPEG), 2000 (JPEG2000)
JBIG	Fax	1995–2000
H.261	Video conferencing	1990
H.262, H.262+	DTV, SDTV	1995, 2000
H.263, H.263+	Videophone	1998,2000
MPEG-1	Video CD	1992
MPEG-2	DTV, SDTV, HDTV, DVD	1995
MPEG-4	Interactive video	2000
MPEG-7	Multimedia content description interface	2001
MPEG-21	Multimedia framework	2002
H.264/ MPEG-4 Part 10	Advance video coding	2003
	Fidelity range extensions (high profile), Studio editing, Post processing, Digital cinema	August, 2004

Chapter 3

HIGH EFFICIENCY VIDEO CODING

3.1 HEVC Background and Development [25], [26], [27], [28]

The standard now known as High Efficiency Video Coding (HEVC) reflects the accumulated experience of about four decades of research and three decades of international standardization for digital video coding technology. Its development was a massive undertaking that dwarfed prior projects in terms of the sheer quantity of engineering effort devoted to its design and standardization. The result is now formally standardized as ITU-T Recommendation H.265 and ISO/IEC International Standard 23008-2 (MPEG-H part 2). The first version of HEVC was completed in January 2013 (with final approval and formal publication following a few months later—specifically, ITU-T formal publication was in June, and ISO/IEC formal publication was in November).

Coding Efficiency of HEVC [19], [20], [21]

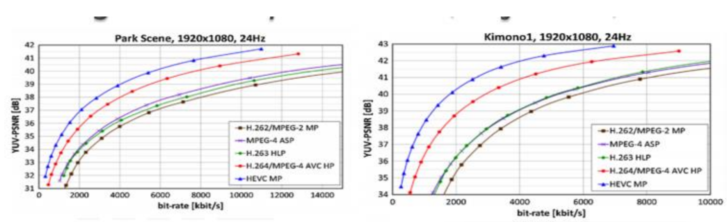


TABLE VI
AVERAGE BIT-RATE SAVINGS FOR EQUAL PSNR FOR ENTERTAINMENT APPLICATIONS

$$PSNR = 10\log_{10}\frac{(2^{bitdepth}-1)^2 * W * H}{\sum_i \{O_i - D_i\}^2}$$

TABLE VI
AVERAGE BIT-RATE SAVINGS FOR EQUAL PSNR FOR ENTERTAINMENT APPLICATIONS

Encoding	Bit-Rate Savings Relative to			
	H.264/MPEG-4 AVC HP	MPEG-4 ASP	H.263 HLP	MPEG-2/ H.262 MP
HEVC MP	35.4%	63.7%	65.1%	70.8%
H.264/MPEG-4 AVC HP	–	44.5%	46.6%	55.4%
MPEG-4 ASP	–	–	3.9%	19.7%
H.263 HLP	–	–	–	16.2%

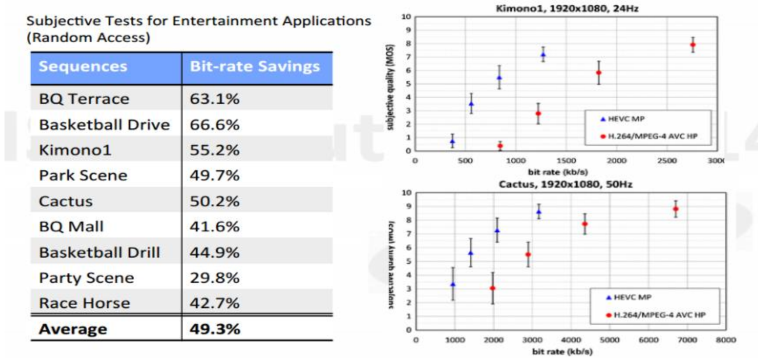
$$PSNR = 10\log_{10}\frac{(2^{bitdepth}-1)^2 * W * H}{}$$

PSNR : Peak Signal t

W = width
H = Height
O_i = Original
D_i = Decoded

Figure 3-1 Comparison of Coding Efficiency of HEVC with other standards [19], [21]

In dB


Subjective Tests for Entertainment Applications (Random Access)

Sequences	Bit-rate Savings
BQ Terrace	63.1%
Basketball Drive	66.6%
Kimono1	55.2%
Park Scene	49.7%
Cactus	50.2%
BQ Mall	41.6%
Basketball Drill	44.9%
Party Scene	29.8%
Race Horse	42.7%
Average	**49.3%**

Figure 3-2 Subjective Coding Efficiency of HEVC [19], [20], [21]

HEVC Key Features [23]

	High Coding Efficiency	High Throughput / Low Power
Larger and Flexible Coding Block Size	X	
More Sophisticated Intra Prediction	X	
Larger Interpolation Filter for Motion Compensation	X	
Larger Transform Size	X	
Parallel Deblocking Filter		X
Sample Adaptive Offset	X	
High Throughput CABAC	X	X
High Level Parallel Tools		X
Parallel Merge/Skip		X

Figure 3-3 Key features of HEVC [23]

3.2 New features of HEVC [19]

- Recursive coding tree structure (64x64 -> 4x4)
- Advanced intra prediction(33 angular , DC ,Planar)
- Greater flexibility in prediction modes and transform block sizes
- DCT based interpolation filter
- Advanced inter prediction and Signaling of modes and motion vectors
- Discrete Sine Transform (DST) for intra(4*4) luma blocks

- Deblocking filter
- Scanning
- Sample adaptive offset

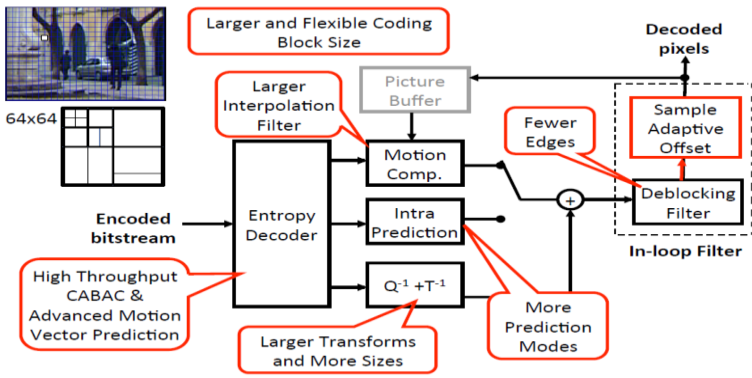


Figure 3-4 New features in HEVC [19]

(AMVP)

INTDCT (4X4), (8X8), (16X16), (32X32)

(Related to DST) (4x4) Intra Luma only

Embedded INTDCT

(4x4), (8x8) and (16x16) INTDCTs are embedded in (32x32) INTDCT

3.3 Working of HEVC in brief

Source video, consisting of sequence of video frames, is encoded or compressed by a video encoder to create a compressed video bit stream. The compressed bit stream is stored or transmitted.

A video decoder decompressed the bit stream to create a sequence of decoded frames.

Steps carried out by video encoder:

Partitioning each picture into multiple units

Predicting each unit using inter or intra prediction, and subtracting the prediction from the unit

Transforming and quantizing the residual (Original picture unit – Prediction)

Entropy Encoding the transform output, prediction information , mode information and headers

Steps carried out by video decoder:

Entropy decoding and extracting the elements of the coded sequence

Rescaling and inverting the transform stage

Predicting each unit and adding the prediction to the output of inverse transform

Reconstructing a decoded video image

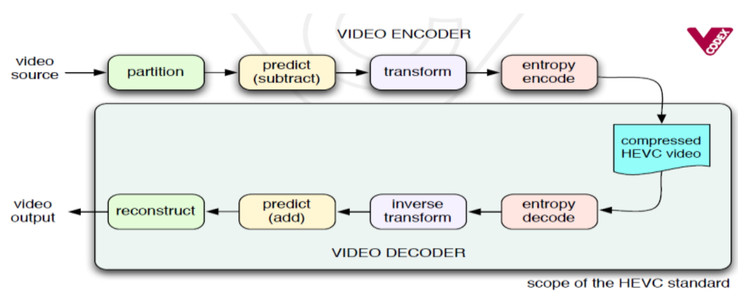


Figure 3-5 Video encoder in HEVC [19]

3.4 HEVC High Level Syntax [25], [32]

An HEVC bitstream consists of a sequence of data units called network abstraction layer (NAL) units. Some NAL units contain parameter sets that carry high-level information regarding the entire coded video sequence or a subset of the pictures within it. Other NAL units carry coded samples in the form of slices that belong to one of the various picture types that are defined in HEVC. Some picture types indicate that the picture can be discarded without affecting the decodability of other pictures, and other picture types indicate positions in the bitstream where random access is possible.

The slices contain information on how decoded pictures are managed, both what previous pictures to keep and in which order they are to be output. Some NAL units contain optional supplementary enhancement information (SEI) that aids the decoding process or may assist in other ways, such as providing hints about how best to display the video. The syntax elements that describe the structure of the bitstream or provide information that applies to multiple pictures or to multiple coded block regions within a picture, such as the parameter sets, reference picture management syntax, and SEI messages, are known as the “high- level syntax” part of HEVC.

A considerable amount of attention has been devoted to the design of the high-level syntax in HEVC, in order to make it broadly applicable, flexible and robust to data losses, and generally highly capable of providing useful information to decoders and receiving systems.

The elements in high level syntax includes:

- NAL Units/Types
- Parameter sets
- Slice Segments/Slices
- Random access
- Reference picture sets

In general, all syntax elements above the slice segment data layer are called high-level synax. These elements have:

- Access to packets.
- Settings of low level coding tools

- Random-access information
- Metadata

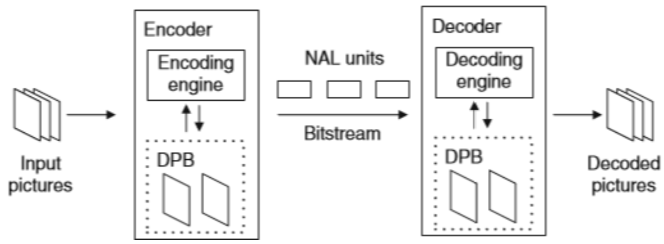


Figure 3-6 Overview of HEVC Encoding and Decoding [25]

3.5 The NAL Unit Header and the HEVC Bitstream [25]

There are two classes of NAL units in HEVC—video coding layer (VCL) NAL units and non-VCL NAL units. Each VCL NAL unit carries one slice segment of coded picture data while the non-VCL NAL units contain control information that typically relates to multiple coded pictures. One coded picture, together with the non-VCL NAL units that are associated with the coded picture, is called an HEVC access unit. There is no requirement that an access unit must contain any non-VCL NAL units, and in some applications such as video conferencing, most access units do not contain non-VCL NAL units. However, since each access unit contains a coded picture, it must consist of one or more VCL NAL units—one for each slice (or slice segment) that the coded picture is partitioned into.

VCL NAL Unit Types [25]

Table 3-1 shows all 32 VCL NAL unit types and their NAL unit type values in the NAL unit header. All VCL NAL units of the same access unit must have the same value of NAL unit type and that value defines the type of the access unit and its coded picture. For example, when all VCL NAL units of an access unit have NAL unit type equal to 21, the access unit is called a CRA access unit and the coded picture is called a CRA picture. There are three basic classes of pictures in HEVC: intra random access point (IRAP) pictures, leading pictures, and trailing pictures.

```
+---------------+---------------+
|0|1|2|3|4|5|6|7|0|1|2|3|4|5|6|7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|  NALType  |  LayerId  | TID |
+-------------+-----------------+
```

Figure 3-7 The two-byte NAL unit header [25]

Table 3-1 The 32 HEVC VCL NAL Unit types [25]

The 32 HEVC VCL NAL unit types

Trailing non-IRAP pictures			
Non-TSA, non-STSA trailing	0	TRAIL_N	Sub-layer non-reference
	1	TRAIL_R	Sub-layer reference
Temporal sub-layer access	2	TSA_N	Sub-layer non-reference
	3	TSA_R	Sub-layer reference
Step-wise temporal sub-layer	4	STSA_N	Sub-layer non-reference
	5	STSA_R	Sub-layer reference
Leading pictures			
Random access decodable	6	RADL_N	Sub-layer non-reference
	7	RADL_R	Sub-layer reference
Random access skipped leading	8	RASL_N	Sub-layer non-reference
	9	RASL_R	Sub-layer reference
Intra random access point (IRAP) pictures			
Broken link access	16	BLA_W_LP	May have leading pictures
	17	BLA_W_RADL	May have RADL leading
	18	BLA_N_LP	Without leading pictures
Instantaneous decoding refresh	19	IDR_W_RADL	May have leading pictures
	20	IDR_N_LP	Without leading pictures
Clean random access	21	CRA	May have leading pictures
Reserved			
Reserved non-IRAP	10–15	RSV	
Reserved IRAP	22–23	RSV	
Reserved non-IRAP	24–31	RSV	

Non-VCL NAL Unit Types [25]

Table 3-2 shows all 32 non-VCL NAL unit types and their NAL unit type values in the NAL unit header.

Table 3-2 The 32 HEVC non-VCL NAL unit types

The 32 HEVC non-VCL NAL unit types

Non-VCL NAL unit types			
Parameter sets	32	VPS_NUT	Video parameter set
	33	SPS_NUT	Sequence parameter set
	34	PPS_NUT	Picture parameter set
Delimiters	35	AUD_NUT	Access unit delimiter
	36	EOS_NUT	End of sequence
	37	EOB_NUT	End of bitstream
Filler data	38	FD_NUT	Filler data
Supplemental enhancement information (SEI)	39	PREFIX_SEI_NUT	
	40	SUFFIX_SEI_NUT	
Reserved	41–47	RSV	
Unspecified	48–63	UNSPEC	

3.6 Parameter Sets

Parameter sets in HEVC are fundamentally similar to the parameter sets in H.264/AVC, and share the same basic design goals—namely bit rate efficiency, error resiliency, and providing systems layer interfaces. There is a hierarchy of parameter sets in HEVC, including the Sequence Parameter Set (SPS) and Picture Parameter Set (PPS) which are similar to their counterparts in AVC. Additionally, HEVC introduces a new type of parameter set called the Video Parameter Set (VPS). Each slice references a single active PPS, SPS and VPS to access information used for decoding the slice.

The PPS contains information which applies to all slices in a picture, and hence all slices in a picture must refer to the same PPS. The slices in different pictures are also allowed to refer to the same PPS. Similarly, the SPS contains information which applies to all pictures in the same coded video sequence.

The VPS contains information which applies to all layers within a coded video sequence, and is intended for use in the upcoming layered extensions of HEVC, which will enable scalable and multiview coding. While the PPS may differ for separate pictures, it is common for many or all pictures in a coded video sequence to refer to the same PPS. Reusing parameter sets is bit rate efficient because it avoids the necessity to send shared information multiple times. It is also loss robust because it allows parameter set content to be carried by some more reliable external communication link or to be repeated frequently within the bitstream to ensure that it will not get lost.

This ability to reuse the content of a picture parameter set in different pictures and to reuse the content of SPSs and VPSs in different CVSs is what primarily distinguishes the concept of a “parameter set” from the “picture header” and “sequence header” syntax used in older standards established prior to AVC.

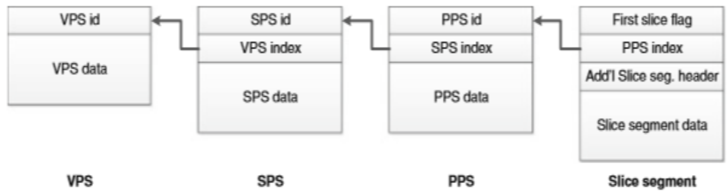


Figure 3-8 Parameter set referencing hierarchy in HEVC [25]

3.7 Block Structures and Parallelism Features in HEVC [25], [24]

The High Efficiency Video Coding (HEVC) standard is designed along the successful principle of block-based hybrid video coding. Following this principle, a picture is first partitioned into blocks and then each block is predicted by using either intra-picture or inter-picture prediction. While the former prediction method uses only decoded samples within the same picture as a reference, the latter uses displaced blocks of already decoded pictures as a reference.

Since inter-picture prediction typically compensates for the motion of real-world objects between pictures of a video sequence, it is also referred to as motion-compensated prediction. While intra-picture prediction exploits the spatial redundancy between neighboring blocks inside a picture, motion-compensated prediction utilizes the large amount of temporal redundancy between pictures.

In either case, the resulting prediction error, which is formed by taking the difference between the original block and its prediction, is transmitted using transform coding, which exploits the spatial redundancy inside a block and consists of a decorrelating linear transform, scalar quantization of the transform coefficients and entropy coding of the resulting transform coefficient levels.

Figure 3-9 shows a block diagram of a block-based hybrid video encoder with some characteristic ingredients of HEVC regarding its novel block partitioning concept.

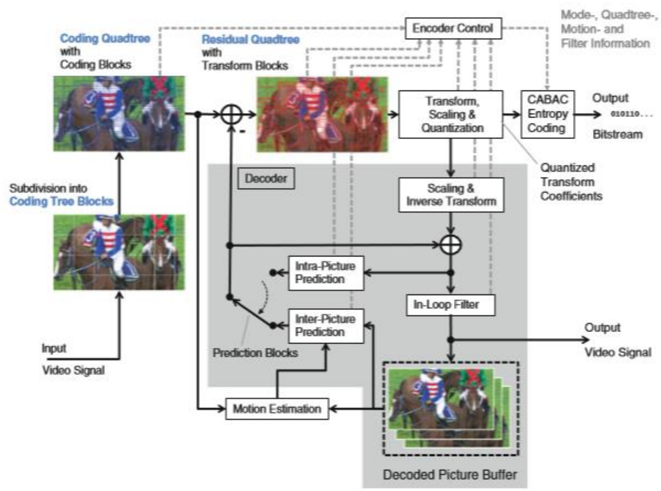


Figure 3-9 Block diagram of an HEVC encoder with built-in decoder (gray shaded)

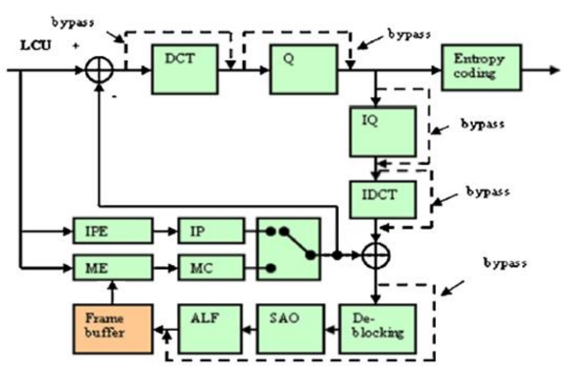


Figure 3-10 HEVC Encoder with lossless encoding mode [24]

This innovative feature of HEVC along with its specific key elements will be one of the main subjects of this chapter. In a first step of this new block partitioning approach, each picture in HEVC is subdivided into disjunct square blocks of the same size, each of which serves as the root of a first block partitioning quadtreestructure,thecodingtree,andwhicharethereforereferredtoascodingtree blocks (CTBs). The CTBs can be further subdivided along the coding tree structure into coding blocks (CBs), which are the entities for which an encoder has to decide between intra-picture and motion-compensated prediction

Parallel picture processing is achieved using:

Slices/Slice segments

- Tiles
- Wavefront Parallel Processing (WPP)

3.8 Picture Partitioning [19], [25]

3.8.1 Coding tree unit:

HEVC has replaced the concept of macro blocks (MBs) with coding tree units. The coding tree unit has a size selected by the encoder and can be larger than the traditional macro blocks. It consists of luma coding tree blocks (CTB) and chroma CTBs. HEVC supports a partitioning of the CTBs into smaller blocks using a tree structure and quad tree-like signaling [10][14].

The quad tree syntax of the CTU specifies the size and positions of its luma and chroma coding blocks (CBs). One luma CB and ordinarily two chroma CBs, together with associated syntax, form a coding unit (CU) for 4:2:0 format.

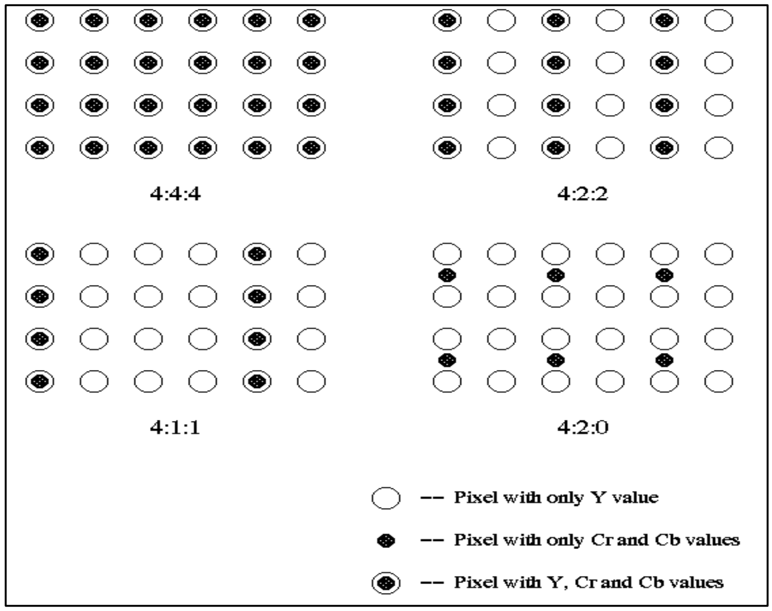


Figure 3-11 Format for YUV components [44]

Each CU has an associated partitioning into prediction units (PUs) and a tree of transform units (TUs). Similarly, each CB is split into prediction blocks (PB) and transform blocks (TB) [15].The decision whether to code a picture area using inter-picture or intra-picture prediction is made at the CU level. Figure 3-12 shows different sizes of a CTU [17].

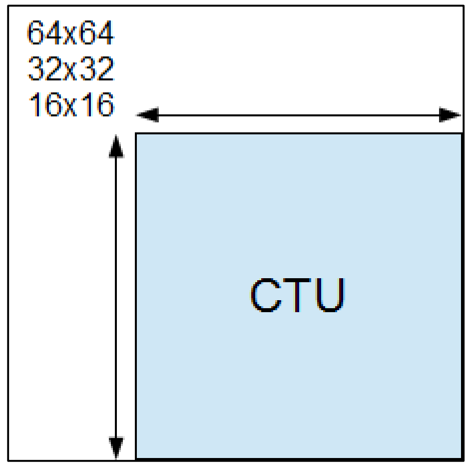


Figure 3-12 Different sizes of CTU [17]

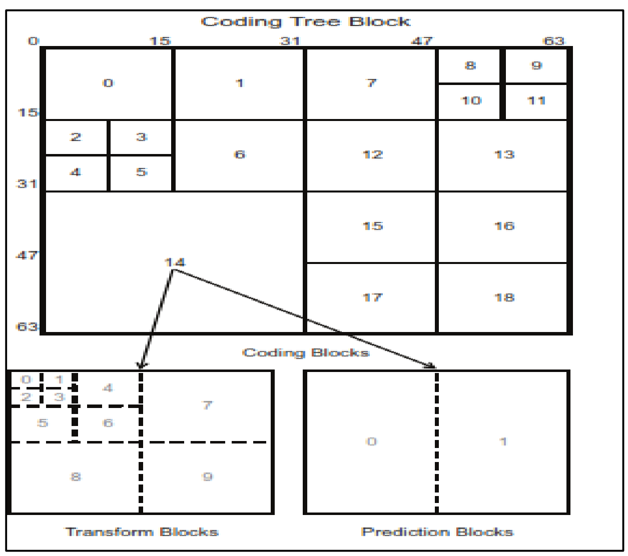


Figure 3-13 Sub-division of a CTB into TBs and PBs [8].

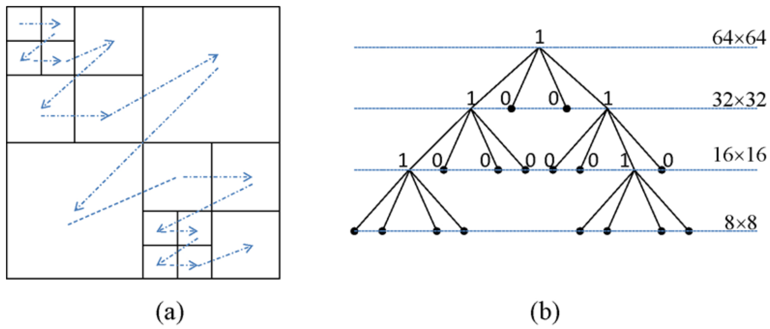


Figure 3-14 Example of CTU, partitioning and processing order [33]

Larger CTU sizes typically enable better compression.

HEVC then supports a partitioning of the CTBs into smaller blocks using a tree structure and quad tree-like signaling.

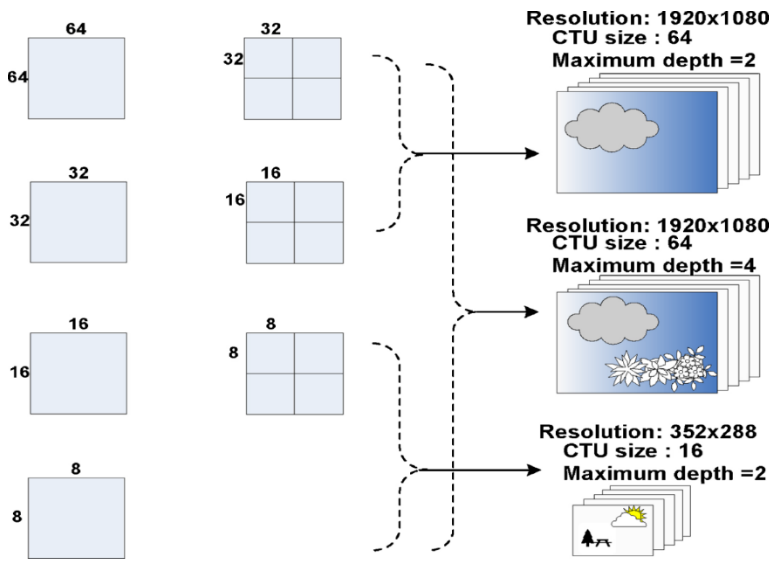


Figure 3-15 Flexible CU Partitioning [33]

3.9 Transform Units [33], [34]

Similar with the PU, one or more TUs are specified for the CU.

HEVC allows a residual block to be split into multiple units recursively to form another quad tree which is analogous to the coding tree for the CU [12].

The TU is a basic representative block having residual for applying the integer transform and quantization.

For each TU, one integer transform having the same size as the TU is applied to obtain residual transform coefficients.

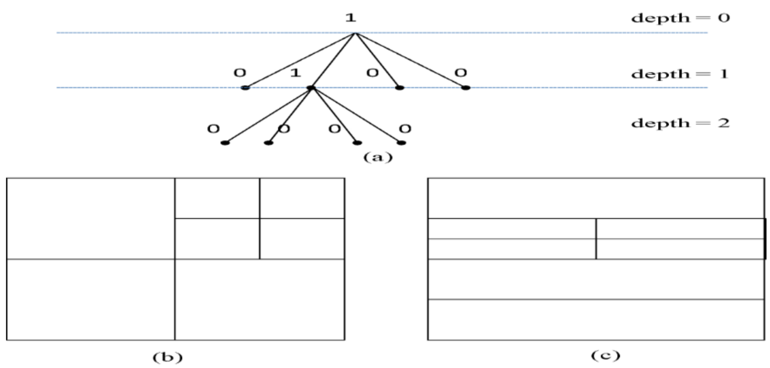


Figure 3-16 Examples of transform tree and block partitioning [33]

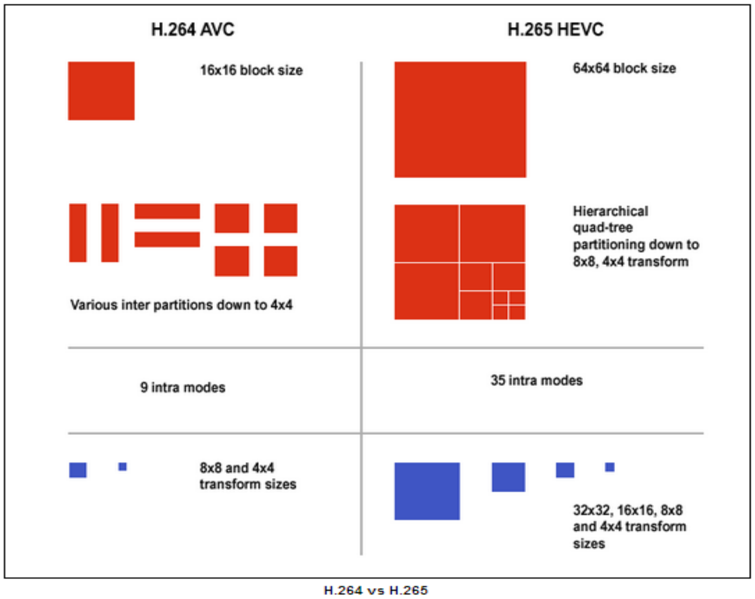


Figure 3-17 Block partitioning comparison between HEVC and H.264 [19]

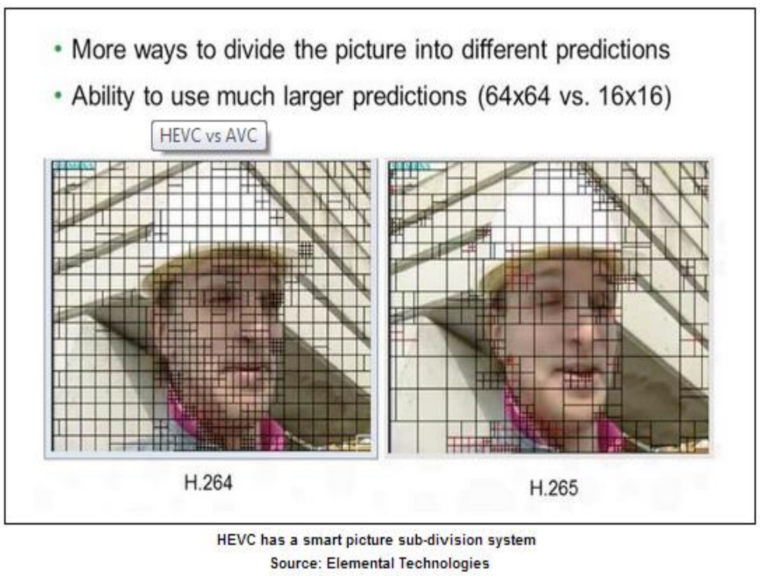


Figure 3-18 Smart picture partition in HEVC compared to H.264 [8]

3.10 Encoder Features:

3.10.1 Motion vector signaling:

The HEVC standard uses a technique called advanced motion vector prediction (AMVP) to derive several most probable candidates based on data from adjacent PBs and the reference picture. A “merge” mode for MV coding can be also used, allowing the inheritance of MVs from neighboring PBs [10]. Moreover, compared to H.264/MPEG-4 AVC, improved “skipped” and “direct” motion inference are also specified [10].

3.10.2 Motion compensation:

The HEVC standard uses quarter-sample precision for the MVs, and for interpolation of fractional-sample positions it uses 7-tap (filter co-efficients: -1, 4, -10, 58, 17, -5, 1) or 8-tap filters (filter co-efficients: -1, 4, -11, 40, 40, -11, 4, 1). In H.264/MPEG-4 AVC there is 6-tap filtering (filter co-efficients: 2, -10, 40, 40, -10, 2) of half-sample positions followed by a bi-linear interpolation of quarter-sample positions [10]. Each PB can transmit one or two motion vectors, resulting either in uni-predictive or bi-predictive coding, respectively [10]. As in H.264/MPEG-4 AVC, a scaling and offset operation may be applied to the prediction signals in a manner known as weighted prediction [10].

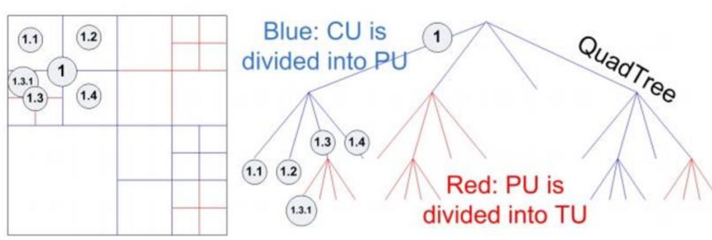


Figure 3-19 Quadtree structure used for motion vectors [35]

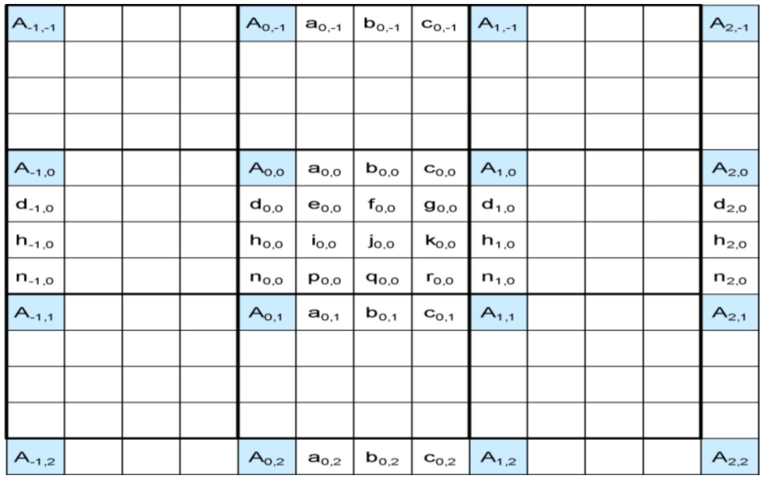


Figure 3-20 Integer and fractional sample positions for luma interpolation [80]

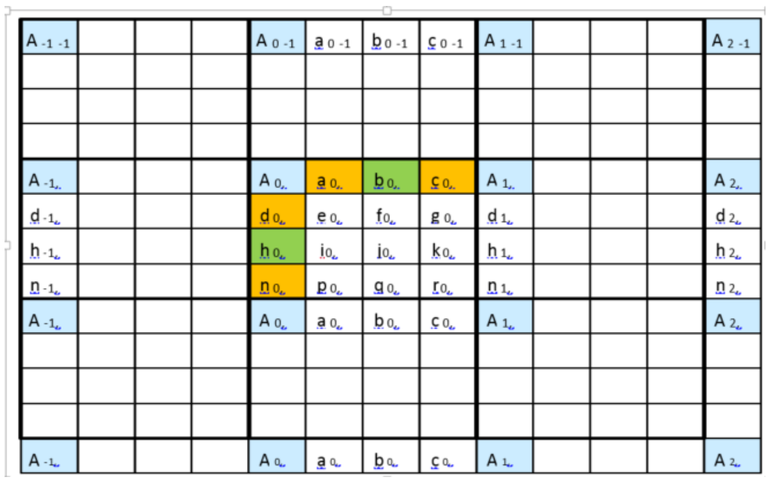


Figure 3-21 Luma Interpolation

	$ha_{0,-1}$	$hb_{0,-1}$	$hc_{0,-1}$	$hd_{0,-1}$	$he_{0,-1}$	$hf_{0,-1}$	$hg_{0,-1}$	$hh_{0,-1}$	
$ah_{-1,0}$	$B_{0,0}$	$ab_{0,0}$	$ac_{0,0}$	$ad_{0,0}$	$ae_{0,0}$	$af_{0,0}$	$ag_{0,0}$	$ah_{0,0}$	$B_{1,0}$
$bh_{-1,0}$	$ba_{0,0}$	$bb_{0,0}$	$bc_{0,0}$	$bd_{0,0}$	$be_{0,0}$	$bf_{0,0}$	$bg_{0,0}$	$bh_{0,0}$	$ba_{1,0}$
$ch_{-1,0}$	$ca_{0,0}$	$cb_{0,0}$	$cc_{0,0}$	$cd_{0,0}$	$ce_{0,0}$	$cf_{0,0}$	$cg_{0,0}$	$ch_{0,0}$	$ca_{1,0}$
$dh_{-1,0}$	$da_{0,0}$	$db_{0,0}$	$dc_{0,0}$	$dd_{0,0}$	$de_{0,0}$	$df_{0,0}$	$dg_{0,0}$	$dh_{0,0}$	$da_{1,0}$
$eh_{-1,0}$	$ea_{0,0}$	$eb_{0,0}$	$ec_{0,0}$	$ed_{0,0}$	$ee_{0,0}$	$ef_{0,0}$	$eg_{0,0}$	$eh_{0,0}$	$ea_{1,0}$
$fh_{-1,0}$	$fa_{0,0}$	$fb_{0,0}$	$fc_{0,0}$	$fd_{0,0}$	$fe_{0,0}$	$ff_{0,0}$	$fg_{0,0}$	$fh_{0,0}$	$fa_{1,0}$
$gh_{-1,0}$	$ga_{0,0}$	$gb_{0,0}$	$gc_{0,0}$	$gd_{0,0}$	$ge_{0,0}$	$gf_{0,0}$	$gg_{0,0}$	$gh_{0,0}$	$ga_{1,0}$
$hh_{-1,0}$	$ha_{0,0}$	$hb_{0,0}$	$hc_{0,0}$	$hd_{0,0}$	$he_{0,0}$	$hf_{0,0}$	$hg_{0,0}$	$hh_{0,0}$	$ha_{1,0}$
	$B_{0,1}$	$ab_{0,1}$	$ac_{0,1}$	$ad_{0,1}$	$ae_{0,1}$	$af_{0,1}$	$ag_{0,1}$	$ah_{0,1}$	$B_{1,1}$

Figure 3-22 Chroma Interpolation

Motion Compensation consists of three steps:

1. Fetch - reference data, padding is applied if reference block outside picture boundaries.
2. Interpolation – for fractional motion vectors (MV)
3. Weighted Prediction

3.11 Intra-picture prediction:

Intra prediction in HEVC is quite similar to H.264/AVC [15]. Samples are predicted from reconstructed samples of neighboring blocks. The mode categories remain identical: DC, plane, horizontal/vertical, and directional; although the nomenclature for H.264's plane and directional modes has changed to planar and angular modes, respectively [15]. For intra prediction, previously decoded boundary samples from adjacent PUs must be used. Directional intra prediction is applied in HEVC, which supports 17 modes for 4x4 block and 34 modes for larger blocks, inclusive of DC mode [18]. Directional intra prediction is based on the assumption that the texture in a region is directional, which means the pixel values will be smooth along a specific direction [18].

The increased number of directions improves the accuracy of intra prediction. However it increases the complexity and increased overhead to signal the mode [18]. With the flexible structure of the HEVC standard, more accurate prediction, and other coding tools, a significant improvement in coding efficiency is achieved over H.264/AVC [18]. HEVC supports various intra coding methods referred to as Intra_Angular, Intra_Planar and Intra_DC. In [16], an evaluation of HEVC coding efficiency compared with H.264/AVC is provided. It shows that the average bit rate saving for random access high efficiency (RA HE) case is 39%, while for all intra high efficiency (Intra HE) case this bit rate saving is 25%, which is also considerable. It seems that further improvement of intra coding efficiency is still desirable. Figure 3.6.2.3.1 shows different intra prediction modes for HEVC [18].

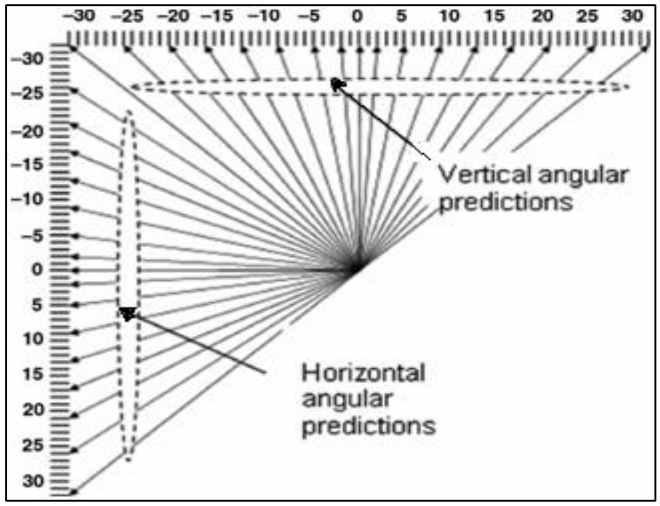


Figure 3-23 Thirty-three Intra prediction modes for HEVC [18]

3.12 Quantization control:

As in H.264/MPEG-4 AVC, uniform reconstruction quantization (URQ) is used in HEVC, with quantization scaling matrices supported for the various transform block sizes [10]. These metrics reflect the HVS.

3.13 Entropy Coding:

HEVC uses context adaptive binary arithmetic coding (CABAC) for entropy coding which is similar to the one used in H.264/MPEG-4 AVC. It has some changes to improve its throughput speed. These improvements can be used for parallel processing architectures and its compression performance, and to reduce its context memory requirements.

4.6 In-loop deblocking filter:

The HEVC standard uses a deblocking filter in the inter-picture prediction loop as used in H.264/MPEG-4 AVC. But design has been simplified in regard to its decision-making and filtering processes, and is made more friendly to parallel processing [10].

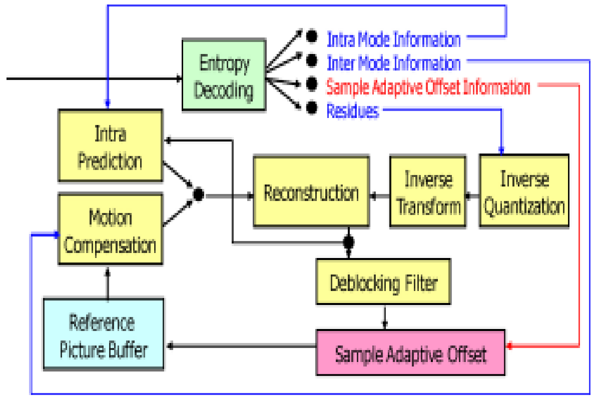


Figure 3-24 Block diagram of deblocking filter [36]

3.14 Sample adaptive offset:

A non-linear amplitude mapping is introduced in the inter-picture prediction loop after the deblocking filter. The goal is to better reconstruct the original signal amplitudes by using a look up table that is described by a few additional parameters that can be determined by histogram analysis at the encoder side [10].

3.15 HEVC Extensions and Emerging Applications [46]:

Range Extensions (Finalized in April 2014)

- Support for 4:2:2 , 4:4:4 color sample video , 12- bit Video

Scalable Video Coding (Finalized in July 2014) (HSVC)

- Supports layered coding -spatial , quality , color gamut scalability

Multiview Video Coding (Finalized in July 2014) (MVC)

-Supports coding of multiple views, 3D stereoscopic video

Screen Content Coding(Expected to be finalized Feb. 2016) (SCC)

-Coding mixed contents consisting of natural video, text / graphics etc.

High dynamic range (HDR) / wide color gamut(WCG)

Post-HEVC activity (VCEG and MPEG AHG work)

Chapter 4

MOTION ESTIMATION IN HEVC

The use of GPUs in video processing and the suitability of the regions of HEVC code in parallel processing is briefed in this chapter. [38]

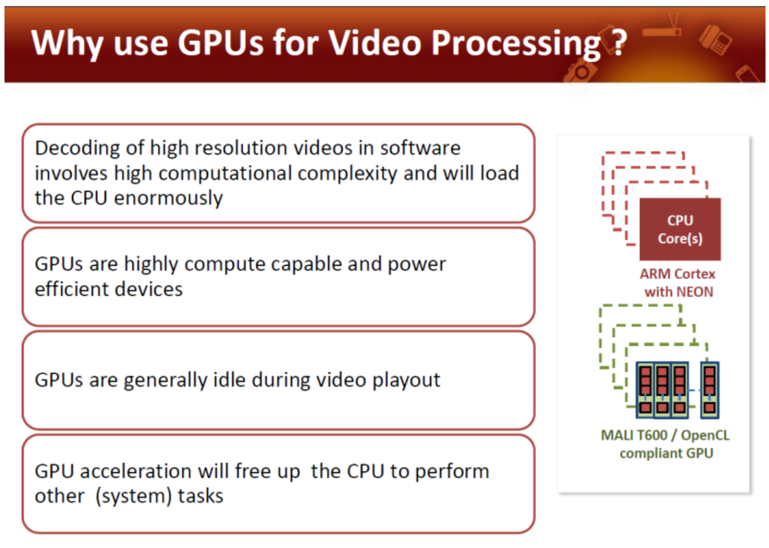


Figure 4-1 Why GPUs?

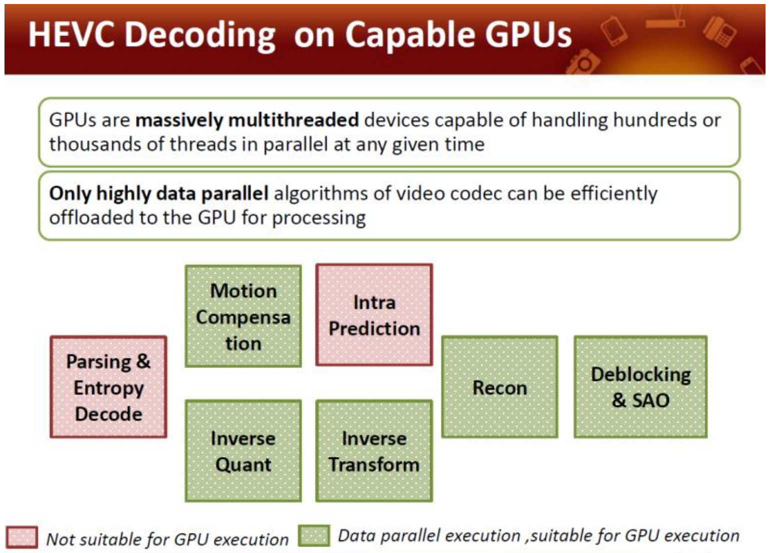


Figure 4-2 Decoding capability of GPUs

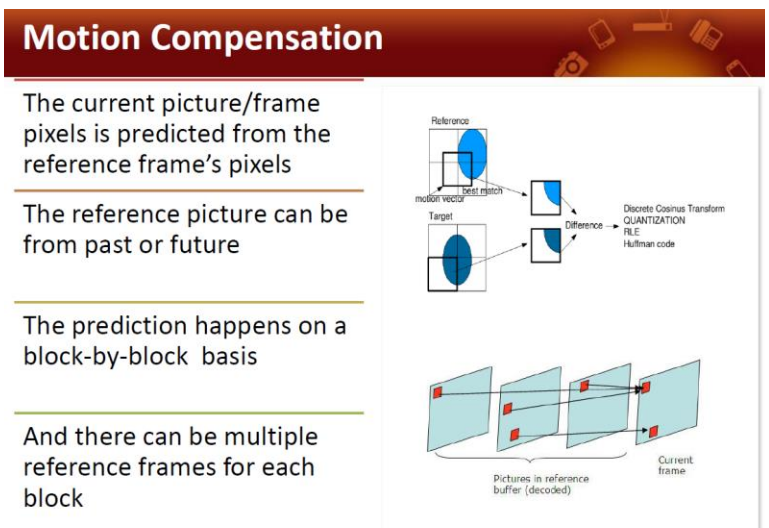


Figure 4-3 Motion Compensation in HEVC

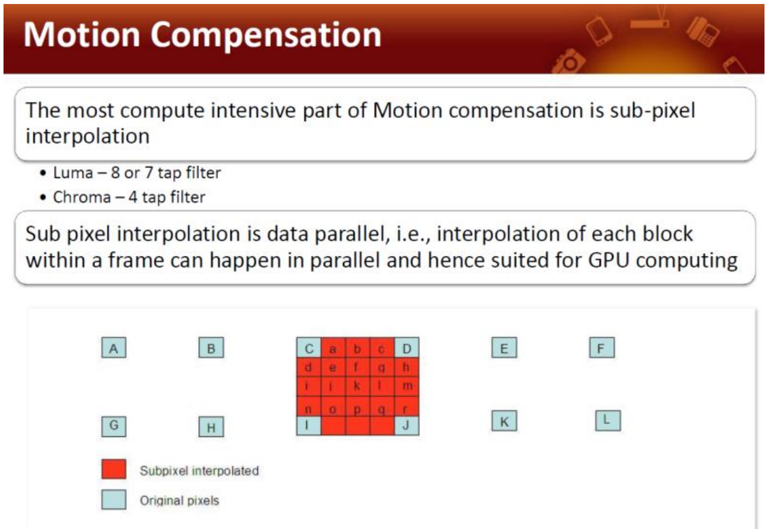


Figure 4-4 Most compute intensive region of Motion Compensation

Chapter 5

PARALLEL COMPUTING USING OPENMP [87]

Parallel computing allows simultaneous execution of threads – not same thing as concurrent execution. Computer Architectures can be classified in two different dimensions, the number of instruction streams that can be processed at any given time, and the number of data streams that can be processed at any given time.

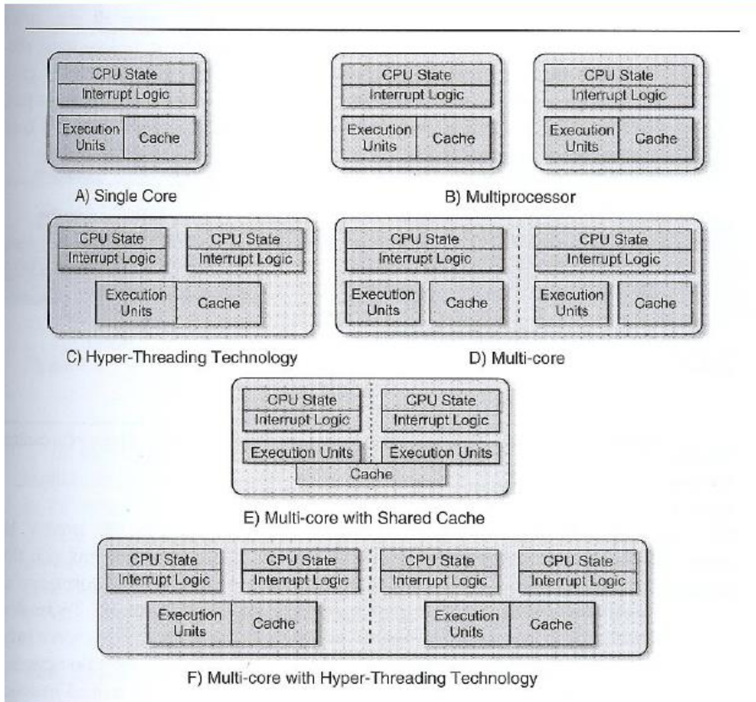


Figure 1.4 Simple Comparison of Single-core, Multi-processor, and Multi-Core Architectures

Figure 5-1 Comparison of different architectures

5.1 Parallel Computing in Microprocessors

Some have thought Moore's law was a predictor of clock speeds, 0.1 MHz – 3.3 GHz.

• Instruction Level Parallelization (ILP) – Out of Order Processing – Hardware Level

•Multiple processes or threads – Software level

–Concurrent thread processing (preemptive)

–simultaneous thread processing (multiple processors)

5.2 Threads

•A Thread is a discrete sequence of related instructions that is executed independently of other instruction sequences.

•Hardware Level Definition: A thread is an execution path that remains independent of other hardware execution paths.

•OS maps software threads to hardware execution

•Thread only needs the architecture state – registers, execution units, etc.

•Logical Processor can be created by duplicating the architecture space.

5.3 What Are Threads Good For?

•Making programs easier to understand

•Overlapping computation and I/O

•Improving responsiveness of GUIs

•Improving performance through parallel execution

5.4 Thread Concurrency vs. Parallelism

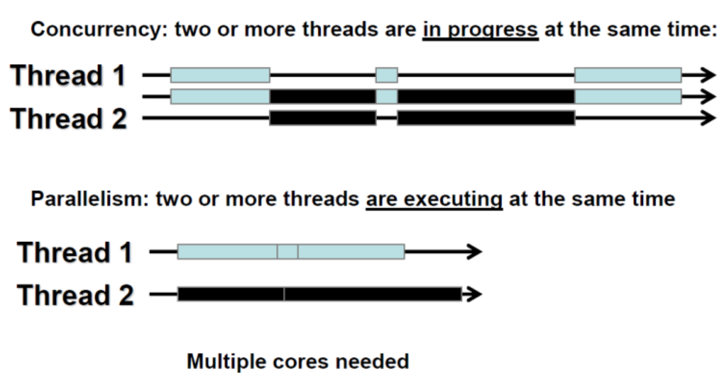


Figure 5-2 Concurrency versus parallelism

5.5 Thread Level Parallelism

•Time-sliced multi-threading – single processor

•Multiple processors – multiple threads or processes run simultaneously on multiple processors

•Physical processor – includes many resources including architecture state (registers, caches, execution units, etc.)

5.6 Hyper-Threading

•Simultaneous multi-threading or SMT - The actual Execution units shared by the different logical processors.

•Intel's implementation called Hyper-threading or HT

•To the OS (e.g., Windows) the computing unit appears as multiple physical processors and threads scheduled accordingly.

•'In the Flynn Taxonomy, a superscalar processor is classified as a MIMD processor (Multiple Instructions, Multiple Data)'

5.7 Speedup Example

Examples: Speedup half the program by 15% using parallel processing, then

Speedup = 1/((1-0.5)+(.5/1.15)) = 1/(.5+.43) = 1.08

Thus whole program speedup by 8 percent.

5.8 Speed Up

Expressing in terms of the serial and parallel portions:

Speedup = 1/(S + (1-S)/n)

Where S is the time spent executing the serial portion of program and n is the number of execution cores

If n = 1, then there is no speedup

As n = increases without bound,

Speedup = 1/S

5.9 Parallel Code vs. Parallel Processors

•For 2 cores and a 30% parallelized program

•1/(.7 + .3/2) = 1.176 or S = 17.6 percent 1/(.7+.3/4) = 1.29 or 29 percent

•1/(0.4+ .6/2) = 1.818 = 82 %

•Thus only when the program is mostly parallelized does adding more processors help the most

Typical Stack Representation for Multithreaded Process

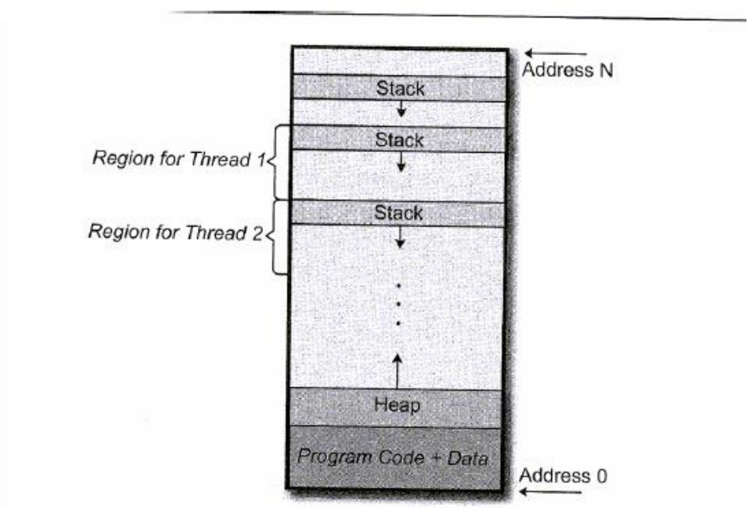


Figure 5-3 Stack representation of Multithreaded process

5.10 More General Threads Model

•When program begins execution, only one user thread, called the main thread, is active

•The main thread can create other threads, which execute other functions

•Created threads can also create additional threads

•How this is done varies according to programming language or API

Operating States of a Thread

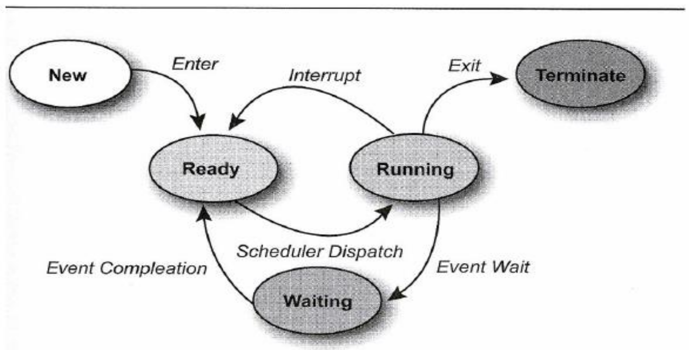


Figure 5-4 Operating states of a thread

5.11 Application Threads

Application threads can be implemented at the application level using established API's such as OpenMP, Pthreads, Windows threads - Win32/MFC, Intel Threads, etc. Examine the OpenMP Program:

```
#include <stdio.h>
#include <omp.h>
int main()
{
int threadID, totalThreads;
/* OpenMP pragma specifies that following block is
going to be parallel and the threadID variable is
private in this openmp block. */
```

```
#pragma omp parallel private(threadID)

{

threadID = omp_get_thread_num();

printf("\nHello World is from thread %d\n",

(int)threadID);

/* Master thread has threadID = 0 */

if (threadID == 0) {

printf("\nMaster thread being called\n");

totalThreads = omp_get_num_threads();

printf("Total number of threads are %d\n",

totalThreads);

}

}

return 0;

}
```

Each Thread Executes The Same Code Unless Directed by IF Statement

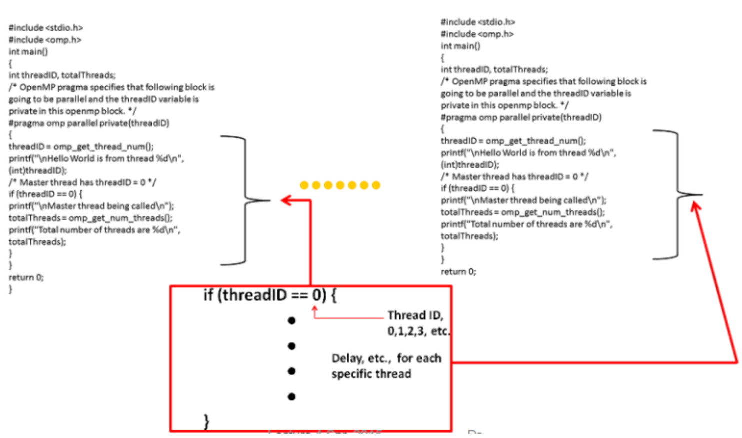


Figure 5-5 Sample openMP program

Example - Find the Number of Processors

Function omp_get_num_procs returns the number of physical processors available to the parallel program

int omp_get_num_procs (void);

Example:

int t;

...

t = omp_get_num_procs();

Get Number of Threads Currently in Use

• omp_get_thread_num();

• Returns the number of threads currently in use

Setting the Number of Threads

• Function omp_set_num_threads allows you to set the number of threads that should be active in parallel sections of code

• void omp_set_num_threads (int t);

• The function can be called with different arguments at different points in the program

• Example:

• int t;

• …

• omp_set_num_threads (t);

5.12 Reductions

•Given associative binary operator $\oplus$ the expression

$$a_1 \oplus a_2 \oplus a_3 \oplus \ldots \oplus a_n$$

is called a reduction

•The 'value'-finding program performs a sum-reduction without specifying a critical section.

```
double area, pi, x;
int i, n;
...
area = 0.0;
#pragma omp parallel for private(x) reduction(+:area)
for (i = 0; i < n; i++) {
x = (i + 0.5)/n;
area += 4.0/(1.0 + x*x);
}
pi = area / n;
```

5.13 OpenMP reduction Clause

•OpenMP provides a reduction clause for the parallel for pragma

•Reduction Eliminates need for:

Creating private variable

Dividing computation into accumulation of local answers that contribute to global result

5.14 Ways of Exploiting Parallelism

•Data decomposition (Domain)

•Task (functional) decomposition

•Pipelining (Data Flow)

5.15 Different Forms of Decomposition

•Task - Different activities assigned to different threads

•Data – Multiple threads performing the same operation but on different blocks of data

•Data Flow – One thread’s output is the input to a second thread

5.16 Parallel Programming Patterns

•Task-level parallelism - Task

In this pattern, the problem is decomposed into a set of tasks that operate independently. It is often necessary remove dependencies between tasks or separate dependencies using replication.

•Divide and Conquer - Task/Data

The problem is divided into a number of parallel sub-problems. Each sub-problem is solved independently.

•Geometric Decomposition - Data

The geometric decomposition pattern is based on the parallelization of the data structures.

•Pipeline - Data Flow

Identical to that of an assembly line. - break down the computation into a series of stages and have each thread work on a different stage simultaneously.

•Wavefront - Data Flow

The wavefront pattern is useful when processing data elements along a diagonal in a two-dimensional grid.

Chapter 6

IMPLEMENTATION

6.1 Analysis and algorithm implementation

JCTVC has provided an open source implementation of the state-of-art video codec, HEVC [74]. The idea behind this thesis can be organized as modules as follows:

6.1.1 Module 1: Analysis of the basic HM software (HM 16.7 is used in this thesis)

Steps:

1. Download the HM16.7 (or any latest version of HM) from the website link given in [74].
2. Build the convenient version in Visual studio. This will generate a .exe file in the bin folder of the HM source.
3. Open the Intel® vTune™ amplifier->Create New Project->Add the link to the executable->Run basic Hotspot analysis.
4. Parameters to be given to the application while running hotspot analysis should be the same as command line parameters that will be given to actually encode the video sequence: -c <path_to_cfg/sample.cfg> - i <path_to_input/input.y4m> -wdt <width_of_input> -hgt <height_of_input> -f <number_of_frames_to_be_encoded> -fr <frame_rate>
5. The results of running the vTune analysis will be the top 5 Hotspots that consume most of the CPU time while running the application.
6. Click on each of these Hotspots to view the exact functions in the code in which they come from.
7. Modify that particular region of the code and re-run the analysis from step 1 to 5.
8. Notice the improvement in the total CPU time.

9. The top hotspots should disappear if the functions are well optimized.

Video sequences have been chosen based on:

1. Complexity or the amount of movement in the video (Easy, Medium, Hard).
2. Resolution (Since HEVC is meant for encoding high resolution streams, 1080p and 2060p were decided to be used for analysis. But 2060p videos took 6 hours to encode even on the most powerful Intel hardware since the HM code is not well optimized)

Table 6-1 Video Sequences used in Intel ® vTune™ amplifier analysis

Name of sequence	Resolution	Complexity
Ducks Take Off	1920x1080 (1080p)	Easy
Park Joy	1920x1080 (1080p)	Medium
Crowd Run	1920x1080 (1080p)	Hard
Ducks Take Off	1280x720 (720p)	Easy
Park Joy	1280x720 (720p)	Medium
Crowd Run	1280x720 (720p)	Hard

Note: All these sequences are downloaded from link given in reference [85]

6.1.2 Module 2: Change the configuration parameters of the HM software

HEVC software provides a wide range of parameters as specified in the HM software manual [74]. Playing around with these parameters will save a lot of encoding time at reasonable/no loss of quality.

In this module, different parameters are changed, the encoding is carried out to see the results and the final best parameter settings for the HM encoder are chosen.

6.1.3 Module 3: vTune analysis of modified code to find parallelizable loops

vTune is a very powerful tool which has the best capabilities of analysis of the code in every aspect. vTune lets us see the loops in the code which take a lot of time of the encoder.

These loops are spotted using the “Functions and Loops” option in the Bottom Up pane of the results from analysis. These loops are checked for parallelism by using Open MP. A detailed and repetitive analysis of the HM code for parallel loops revealed that the code is not well suited for parallelism, since parallelizing degraded the performance badly.

There are lots of loops in the code which have already been optimized using vectorization. Memory misaligned functions/loops were also spotted and analyzed that proper memory alignment of these will lead to less cache misses and hence improved performance at the microprocessor architecture level.

6.1.4 Module 4: Performance comparison of Original and Optimized HM encoders

Finally after all the analysis until module 3, the .exe files from both original and optimized code are run for the following setting:

Table 6-2 Encoder Comparison Configurations used in this thesis

Parameters tested for	Parameter value	Number of iterations
Quantization Parameter (QP)	22,24,26,28,30,32	6
Profile (Main)	Main	1
Resolution	1080p, 720p	2
Videos used	ParkJoy, CrowdRun, DucksTakeOff	3
Encoder Versions compared	Original and Optimized	2
Total number of iterations		(6*1*2*3*2)=72

6.2 Metrics used for comparison:

Each of the 72 iterations will be evaluated for the following metrices:

1. PSNR
2. Encoding Time
3. RD-plot

6.3 Experimental Setup

The following include the configuration and requirements for carrying out the thesis:

6.3.1 System:

CPU: Intel ® Core ™ i7-4770R CPU @ 3.20GHz

GPU: Intel® Iris™ Pro Graphics 5200

6.3.2 Software:

HM16.7 reference software

6.3.3 Tools/IDEs:

Microsoft Visual Studio

Intel ® vTune ™ amplifier

Matlab

6.3.4 Test Sequences:

Crowd Run

Park Joy

Ducks Take off

Chapter 7

Measurement Methods and Results

7.1 Measurement Quality Metrics Used for Comparison

BD-rate and BD-PSNR [47]

The program below computes the Bjontegaard metric to measure the average difference between two rate-distortion curves:

```
function avg_diff = bjontegaard(R1,PSNR1,R2,PSNR2,mode)


%BJONTEGAARD    Bjontegaard metric calculation
%   Bjontegaard's metric allows to compute the average gain in PSNR or the
%   average per cent saving in bitrate between two rate-distortion
%   curves [1].
%   Differently from the avsnr software package or VCEG Excel [2] plugin this
%   tool enables Bjontegaard's metric computation also with more than 4 RD
%   points.
%
%   R1,PSNR1 - RD points for curve 1
%   R2,PSNR2 - RD points for curve 2
%   mode -
%       'dsnr' - average PSNR difference
%       'rate' - percentage of bitrate saving between data set 1 and
%                data set 2
%
%   avg_diff - the calculated Bjontegaard metric ('dsnr' or 'rate')
%
```

```
%   (c) 2010 Giuseppe Valenzise
%
%   References:
%
%   [1] G. Bjontegaard, Calculation of average PSNR differences between
%       RD-curves (VCEG-M33)
%   [2] S. Pateux, J. Jung, An excel add-in for computing Bjontegaard metric and
%       its evolution

% convert rates in logarithmic units
lR1 = log(R1);
lR2 = log(R2);

switch lower(mode)
    case 'dsnr'
        % PSNR method
        p1 = polyfit(lR1,PSNR1,3);
        p2 = polyfit(lR2,PSNR2,3);

        % integration interval
        min_int = min([lR1; lR2]);
        max_int = max([lR1; lR2]);

        % find integral
        p_int1 = polyint(p1);
```

```
    p_int2 = polyint(p2);

    int1 = polyval(p_int1, max_int) - polyval(p_int1, min_int);
    int2 = polyval(p_int2, max_int) - polyval(p_int2, min_int);

    % find avg diff
    avg_diff = (int2-int1)/(max_int-min_int);

case 'rate'
    % rate method
    p1 = polyfit(PSNR1,lR1,3);
    p2 = polyfit(PSNR2,lR2,3);

    % integration interval
    min_int = min([PSNR1; PSNR2]);
    max_int = max([PSNR1; PSNR2]);

    % find integral
    p_int1 = polyint(p1);
    p_int2 = polyint(p2);

    int1 = polyval(p_int1, max_int) - polyval(p_int1, min_int);
    int2 = polyval(p_int2, max_int) - polyval(p_int2, min_int);

    % find avg diff
```

```
    avg_exp_diff = (int2-int1)/(max_int-min_int);
    avg_diff = (exp(avg_exp_diff)-1)*100;
end
```

Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR)

The term peak signal-to-noise ratio (PSNR) is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. Because many signals have a very wide dynamic range, (ratio between the largest and smallest possible values of a changeable quantity) the PSNR is usually expressed in terms of the logarithmic decibel scale.

Image enhancement or improving the visual quality of a digital image can be subjective. Saying that one method provides a better quality image could vary from person to person. For this reason, it is necessary to establish quantitative/empirical measures to compare the effects of image enhancement algorithms on image quality.

Using the same set of tests images, different image enhancement algorithms can be compared systematically to identify whether a particular algorithm produces better results. The metric under investigation is the peak-signal-to-noise ratio. If we can show that an algorithm or set of algorithms can enhance a degraded known image to more closely resemble the original, then we can more accurately conclude that it is a better algorithm.

For the following implementation, let us assume we are dealing with a standard 2D array of data or matrix. The dimensions of the correct image matrix and the dimensions of the degraded image matrix must be identical.

The mathematical representation of the PSNR is as follows:

$$PSNR = 20\log_{10}\left(\frac{MAX_f}{\sqrt{MSE}}\right)$$

Figure 7-1 Peak Signal-to-Noise Equation

where the MSE (Mean Squared Error) is:

$$MSE = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(i,j) - g(i,j)\|^2$$

Figure 7-2 Mean Squared Error Equation

This can also be represented in a text based format as:

MSE = (1/(m*n))*sum(sum((f-g).^2))

PSNR = 20*log(max(max(f)))/((MSE)^0.5)

Legend:

f represents the matrix data of our original image

g represents the matrix data of our degraded image in question

m represents the numbers of rows of pixels of the images and i represents the index of that row

n represents the number of columns of pixels of the image and j represents the index of that column

MAXf is the maximum signal value that exists in our original “known to be good” image

The mean squared error (MSE) for our practical purposes allows us to compare the “true” pixel values of our original image to our degraded image. The MSE represents the average of the squares of the "errors" between our actual image and our noisy image.

The error is the amount by which the values of the original image differ from the degraded image.

The proposal is that the higher the PSNR, the better degraded image has been reconstructed to match the original image and the better the reconstructive algorithm. This would occur because we wish to minimize the MSE between images with respect the maximum signal value of the image.

When you try to compute the MSE between two identical images, the value will be zero and hence the PSNR will be undefined (division by zero). The main limitation of this metric is that it relies strictly on numeric comparison and does not actually take into account any level of biological factors of the human vision system such as the structural similarity index. (SSIM)

For color images, the MSE is taken over all pixels values of each individual channel and is averaged with the number of color channels. Another option may be to simply perform the PSNR over a converted luminance or grayscale channel as the eye is generally four times more susceptible to luminance changes as opposed to changes in chrominance. This approximation is left up to the experimenter.

7.2 Results

7.2.1 Initial vTune anaylsis

Settings: HM16.7 code analysed in vTune for DucksTakeOff, CrowdRun and ParkJoy.y4m sequences. Visual Studio is used to build the code in debug mode (before and after optimization) by enabling the settings:

C/C++->Optimization->Inline Functions->No Debugging->Yes

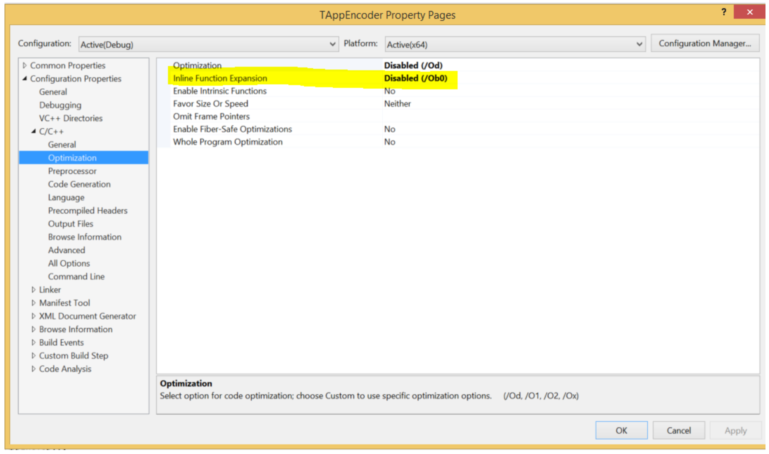


Figure 7-3 Disable Inline function in Visual Studio project property

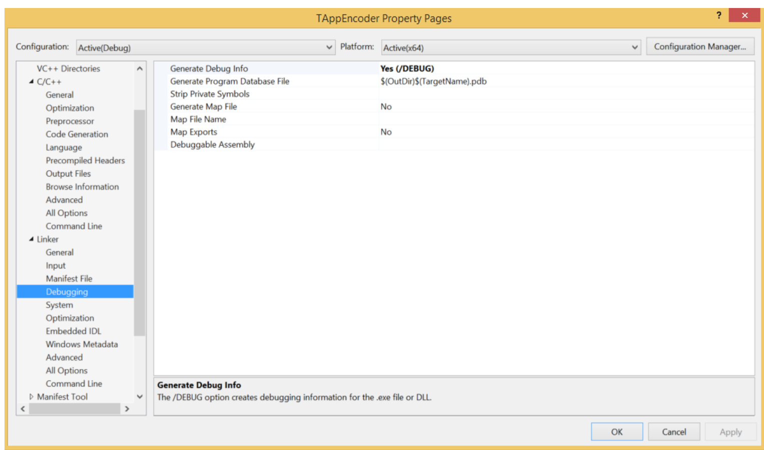


Figure 7-4 Enable debugging in project properties in Visual Studio

Project->Properties->x64(my hardware's configuration-best suited for vTune analysis)

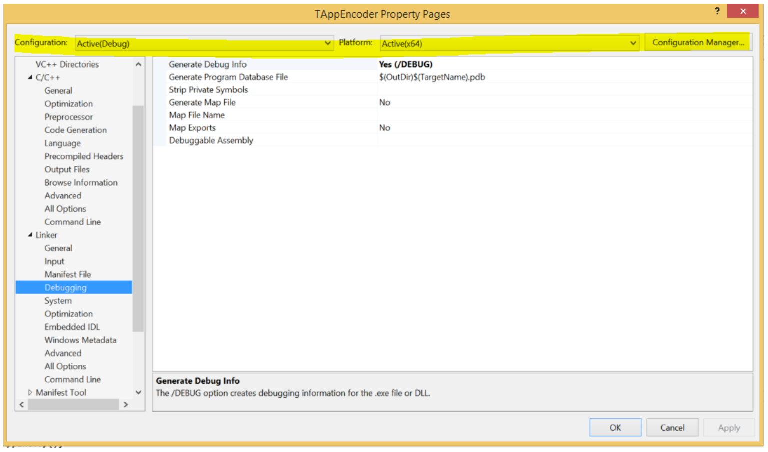


Figure 7-5 Set the configuration to 64 bit in Visual studio project properties

After setting these options, build the HM16.7 project in VS201x.

Steps to run vTune analysis shown below:

Create New Project in vtune amplifier as shown below:

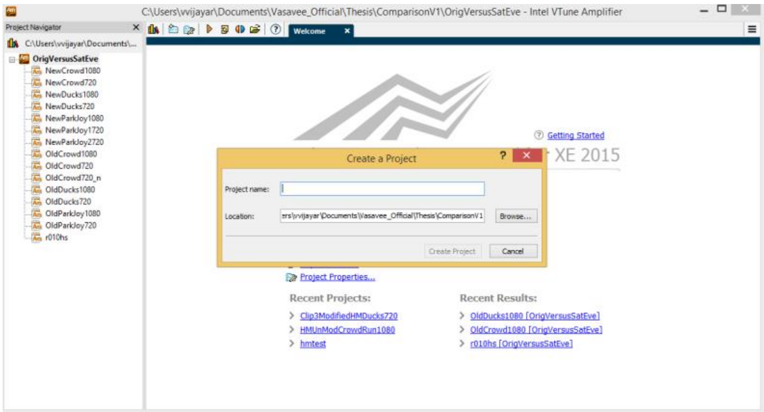


Figure 7-6 Create a new project in Intel ® vTune™ Amplifier

Click on Basic Hotspot analysis:

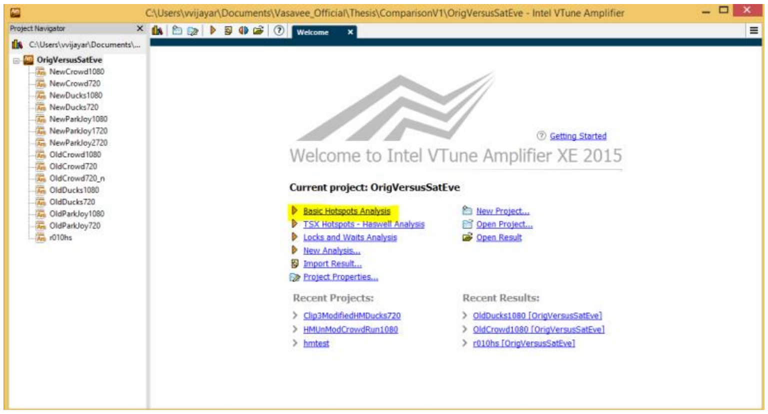


Figure 7-7 Begin a basic hotspot analysis

Click on Project Properties and edit as per requirement:

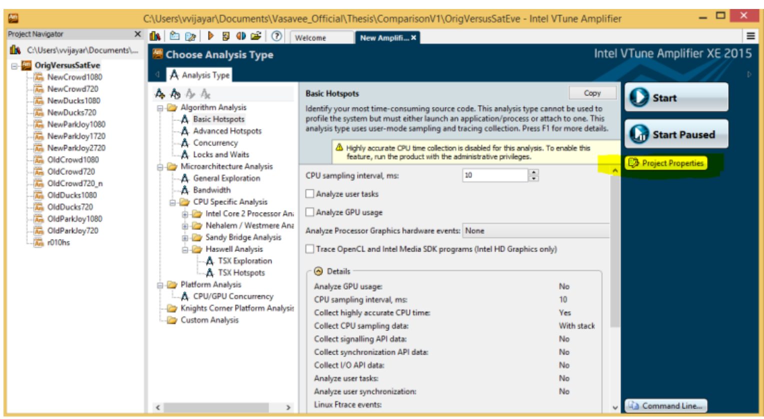


Figure 7-8 Modify the project properties

Enter the application parameters as shown below:

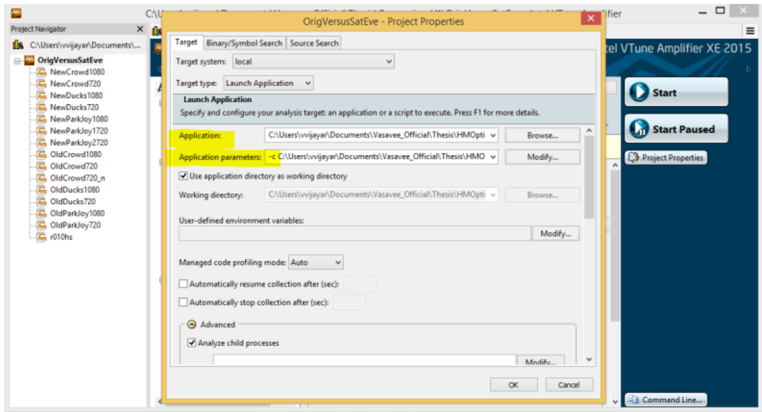


Figure 7-9 Type in the application name and application parameters

Sample application parameters: -c

C:\Users\vvijayar\Documents\Vasavee_Official\Thesis\HMOptimizedV1\cfg\encoder_intra

_main.cfg -i ducks_take_off_1080p50.y4m -hgt 1920 -wdt 1080 -f 10 -fr 30

Click on Start to start the hotspot analysis:

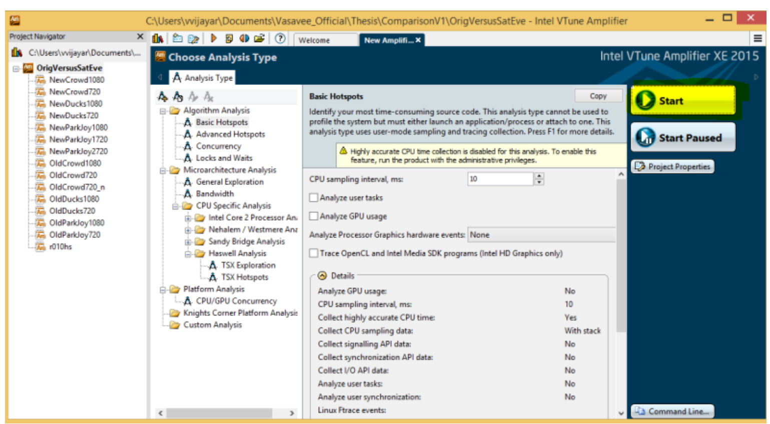


Figure 7-10 Start the analysis

Summary of hotspot analysis is shown as below:

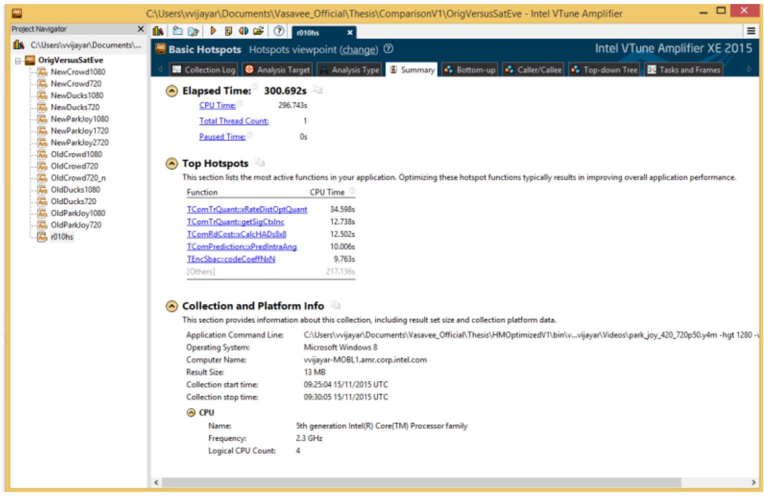


Figure 7-11 Summary of hotspot analysis

Results for 1080p sequences:

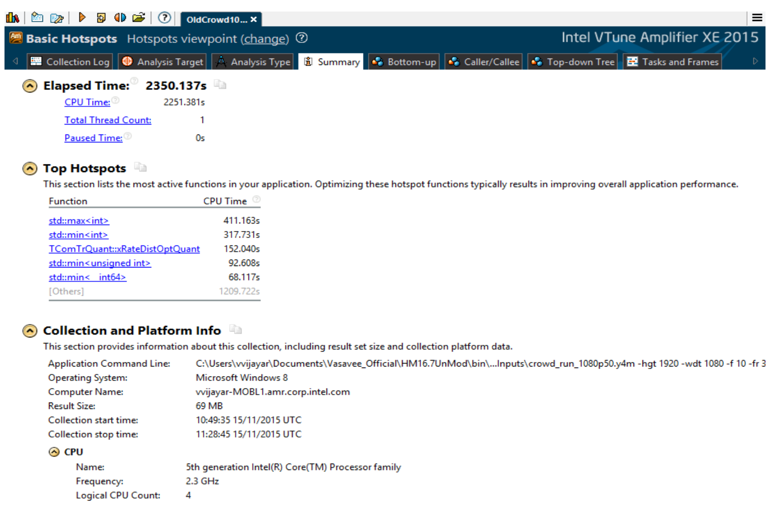


Figure 7-12 Hotspot analysis summary for CrowdRun (Original HM)

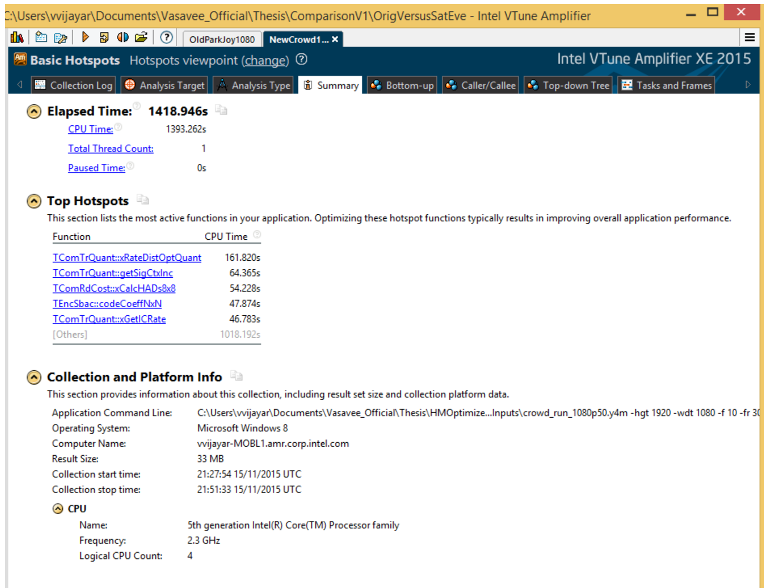


Figure 7-13 Hotspot analysis summary for CrowdRun (Optimized HM)

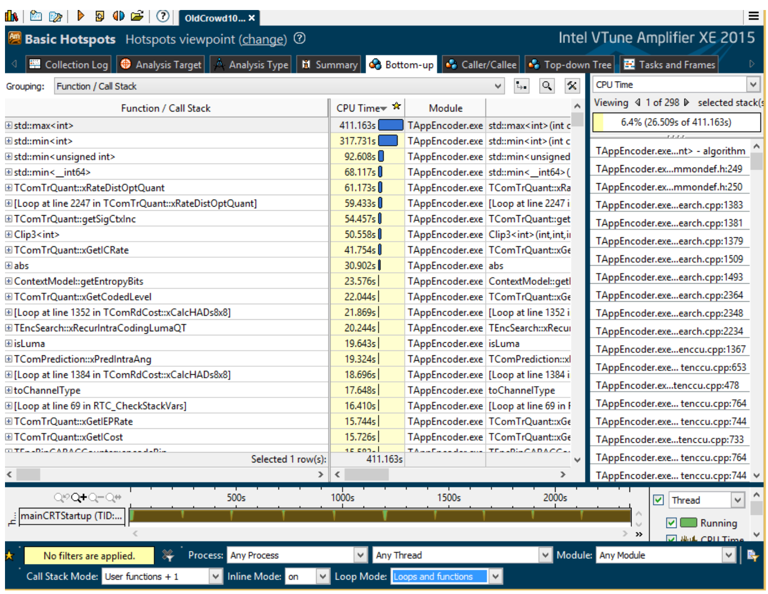


Figure 7-14 Hotspot analysis bottom-up for CrowdRun (Original HM)

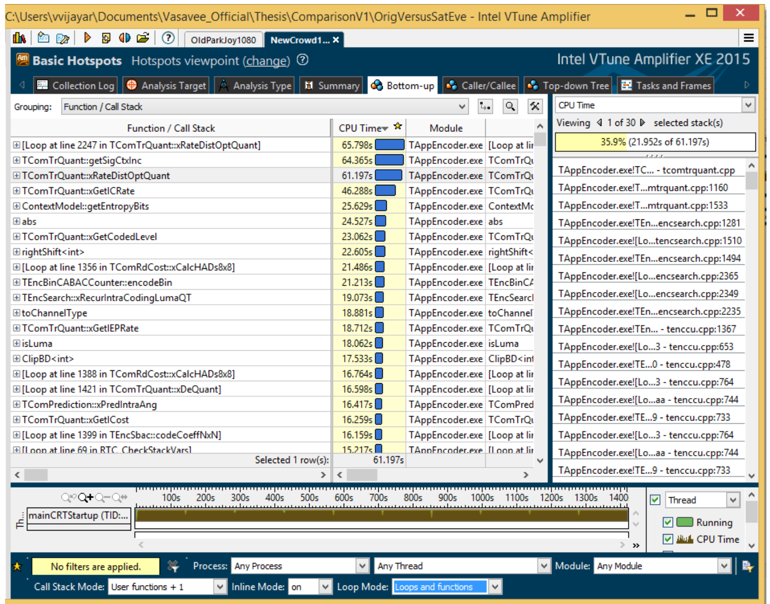


Figure 7-15 Hotspot analysis bottom-up for CrowdRun (Optimized HM)

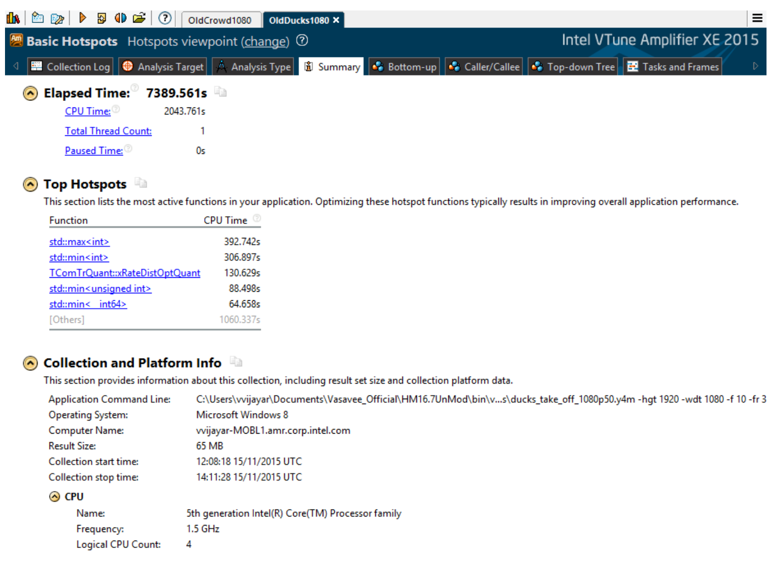


Figure 7-16 Hotspot analysis summary for DucksTakeOff (Original HM)

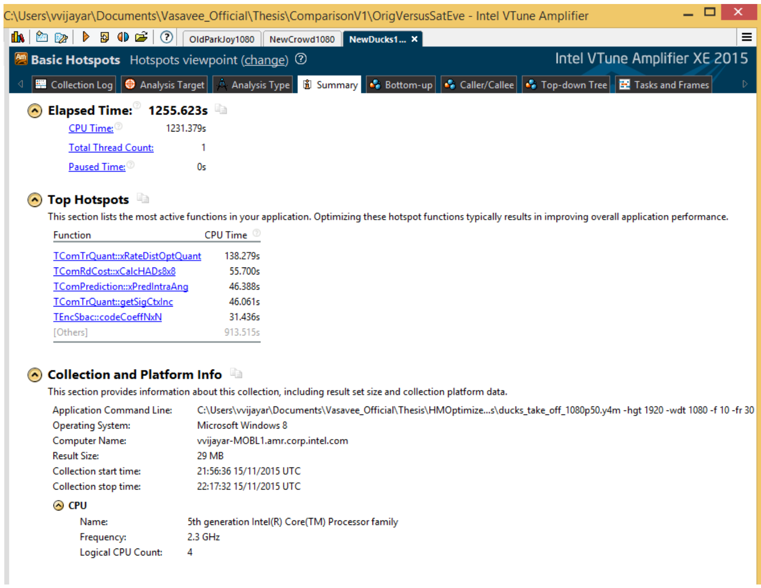


Figure 7-17 Hotspot analysis summary for DucksTakeOff (Optimized HM)

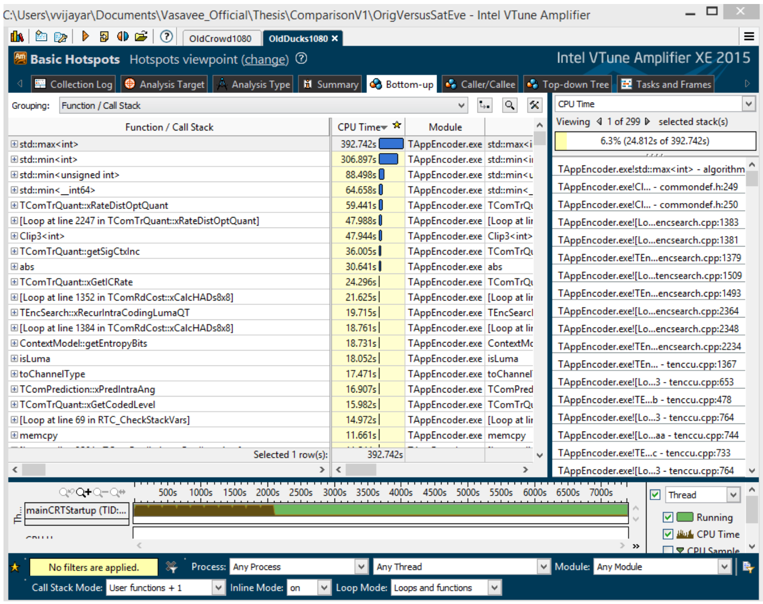


Figure 7-18 Hotspot analysis bottom-up for DucksTakeOff (Original HM)

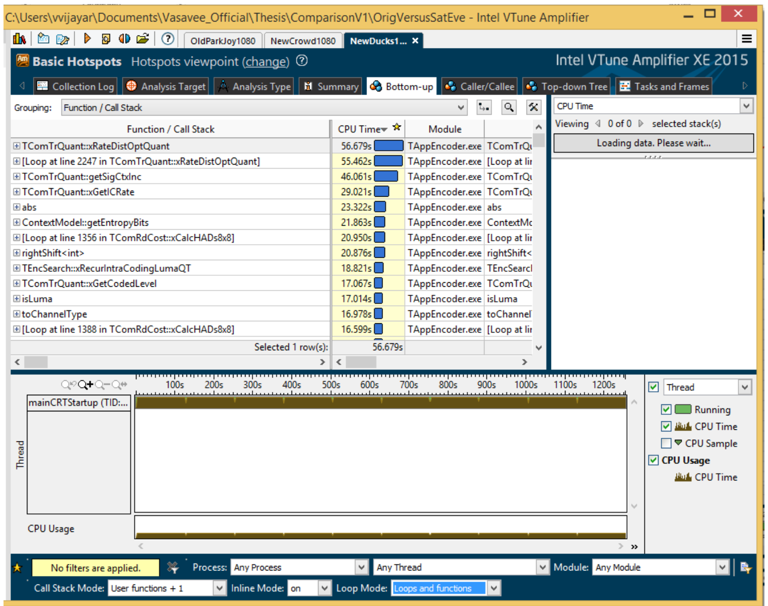


Figure 7-19 Hotspot analysis bottom-up for DucksTakeOff (Optimized HM)

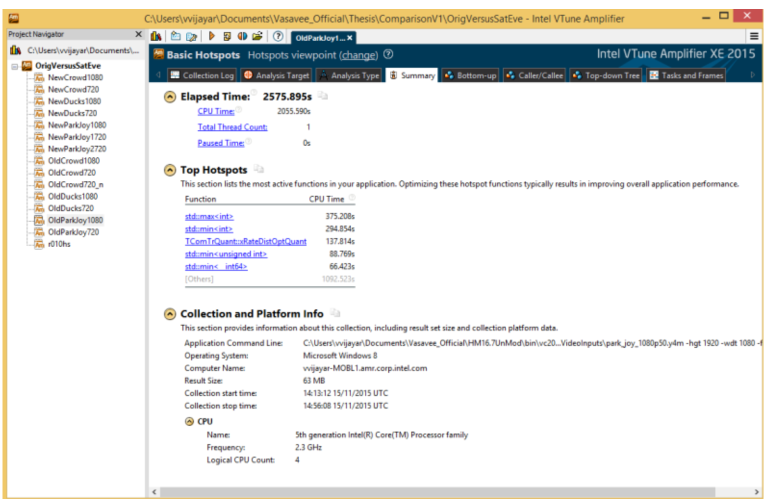


Figure 7-20 Hotspot analysis summary for ParkJoy (Original HM)

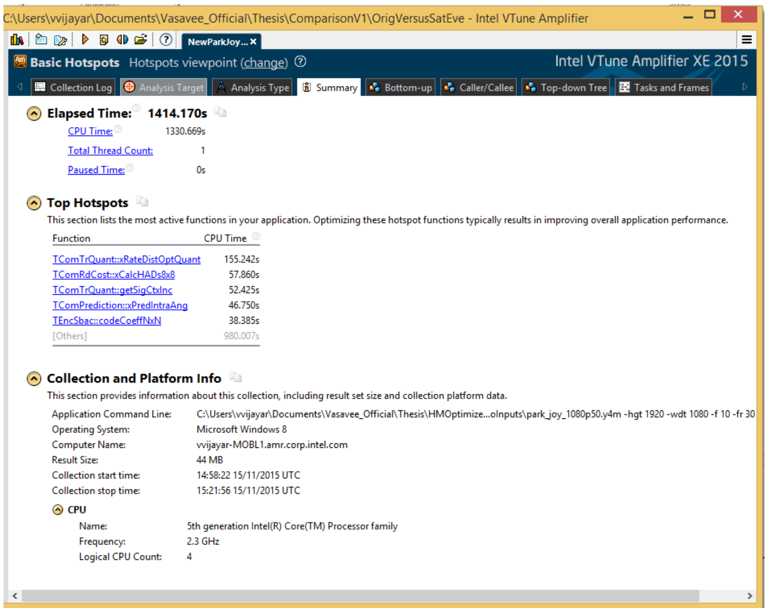


Figure 7-21 Hotspot analysis summary for ParkJoy (Optimized HM)

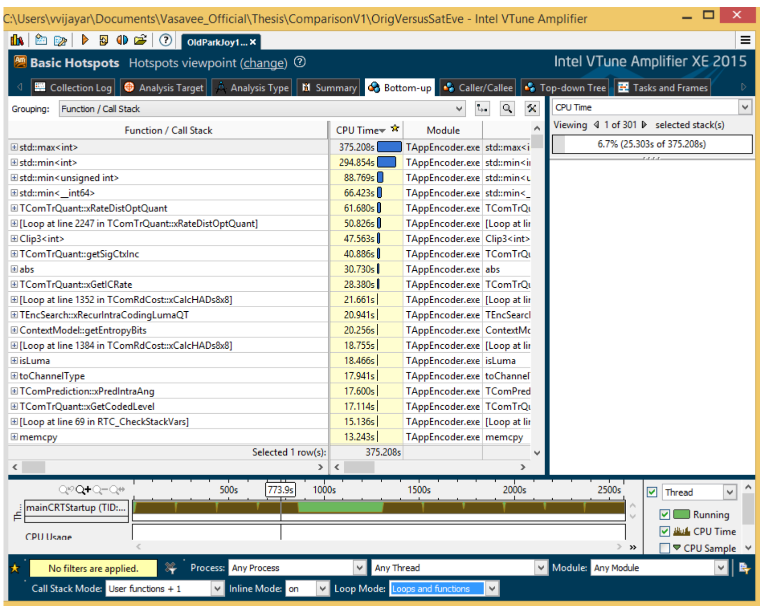


Figure 7-22 Hotspot analysis bottom-up for ParkJoy (Original HM)

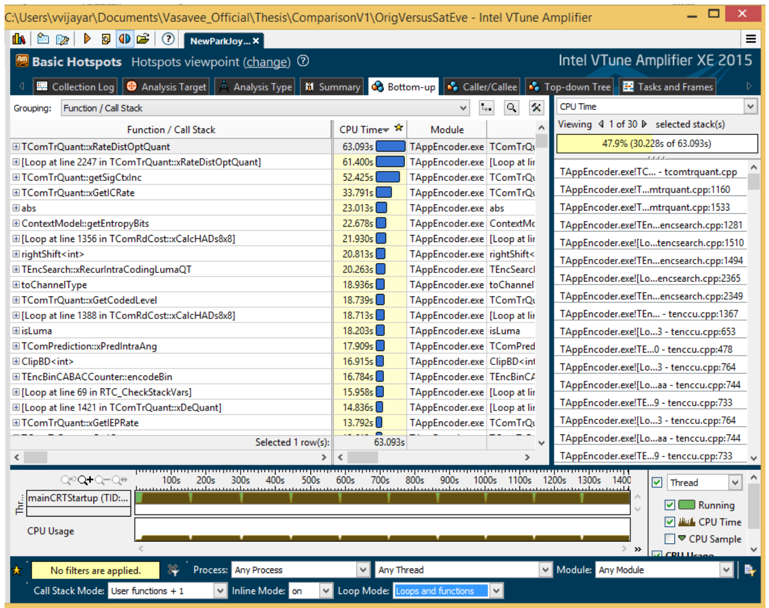


Figure 7-23 Hotspot analysis bottom-up for ParkJoy (Optimized HM)

Results for 720p sequences:

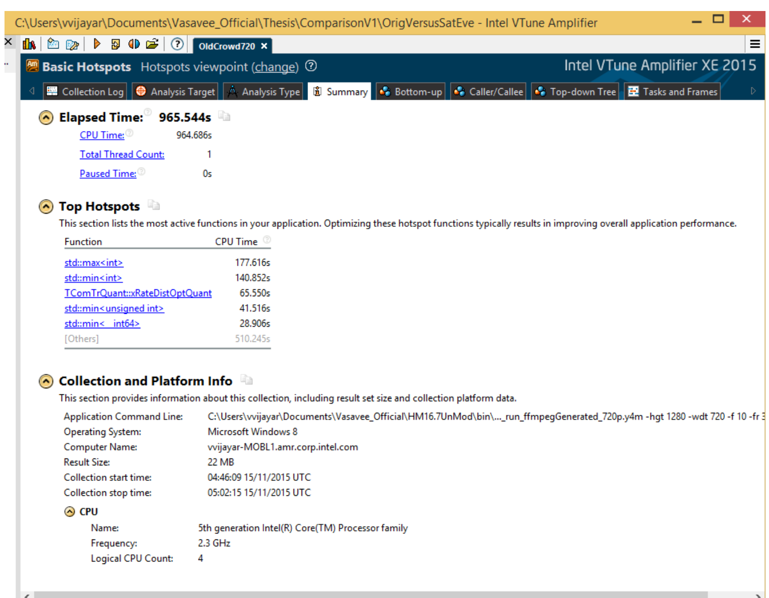


Figure 7-24 Hotspot analysis summary for CrowdRun (Original HM)

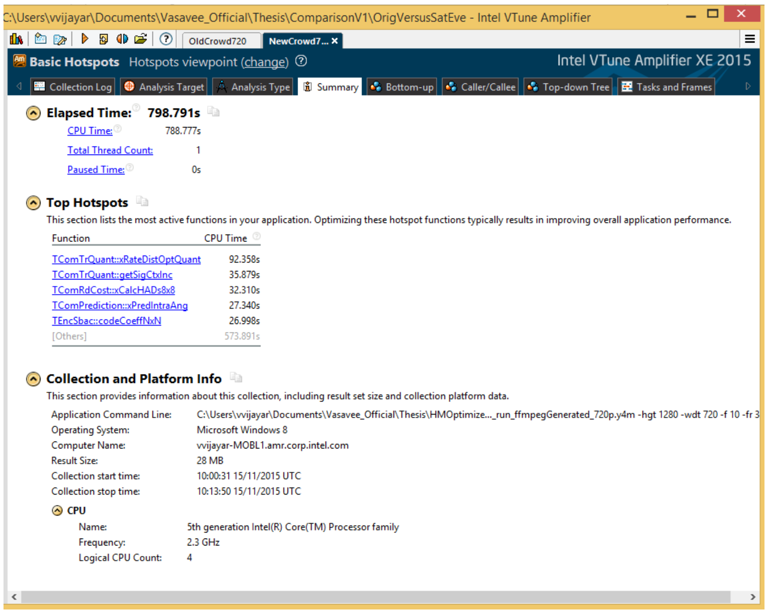


Figure 7-25 Hotspot analysis summary for CrowdRun (Optimized HM)

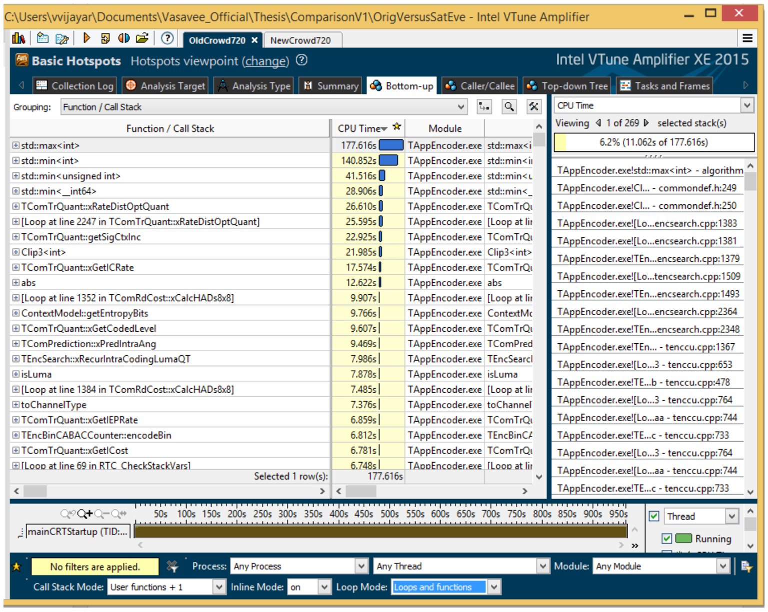


Figure 7-26 Hotspot analysis bottom-up for CrowdRun (Original HM)

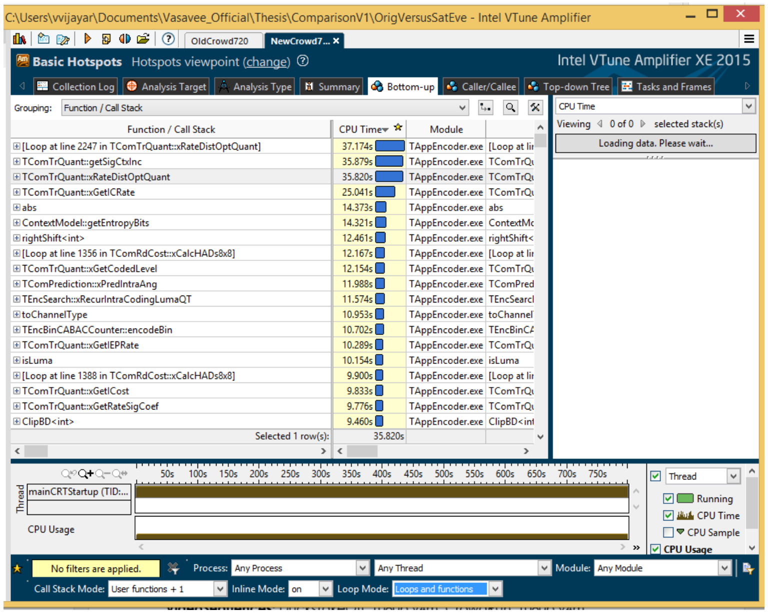


Figure 7-27 Hotspot analysis bottom-up for CrowdRun (Optimized HM)

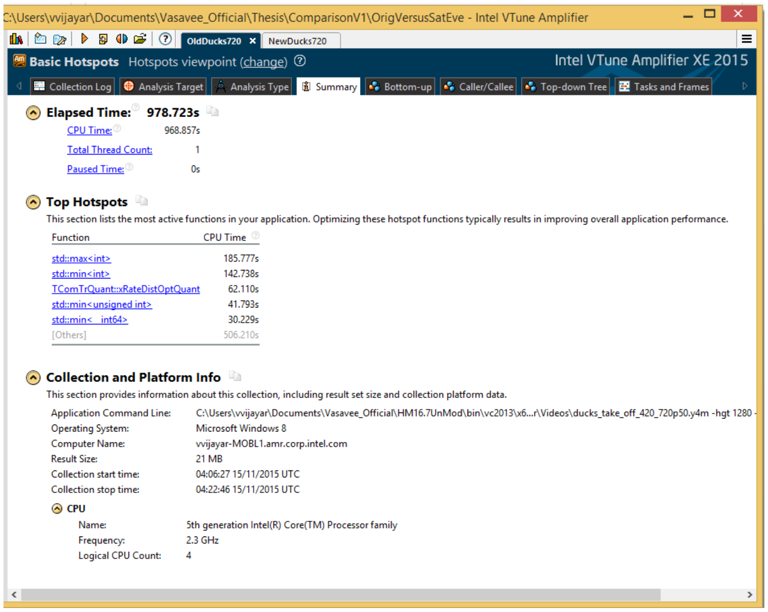


Figure 7-28 Hotspot analysis summary for DucksTakeOff (Original HM)

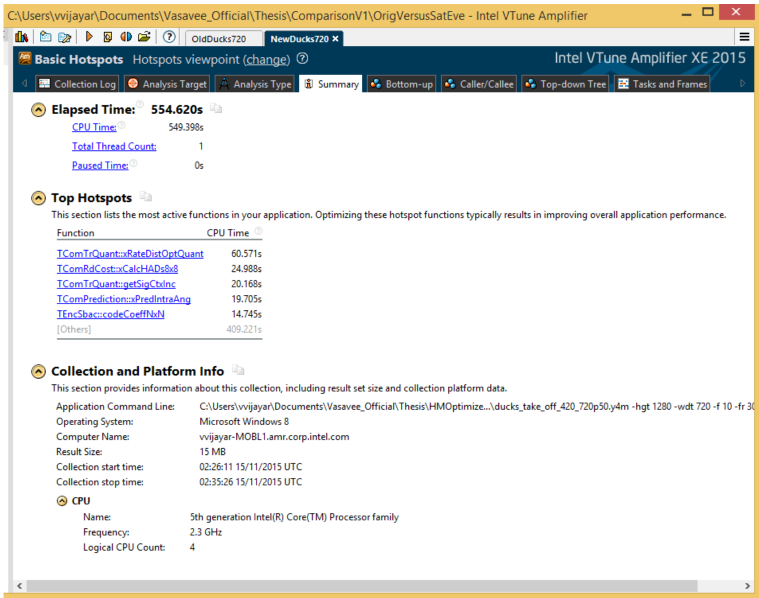


Figure 7-29 Hotspot analysis summary for DucksTakeOff (Optimized HM)

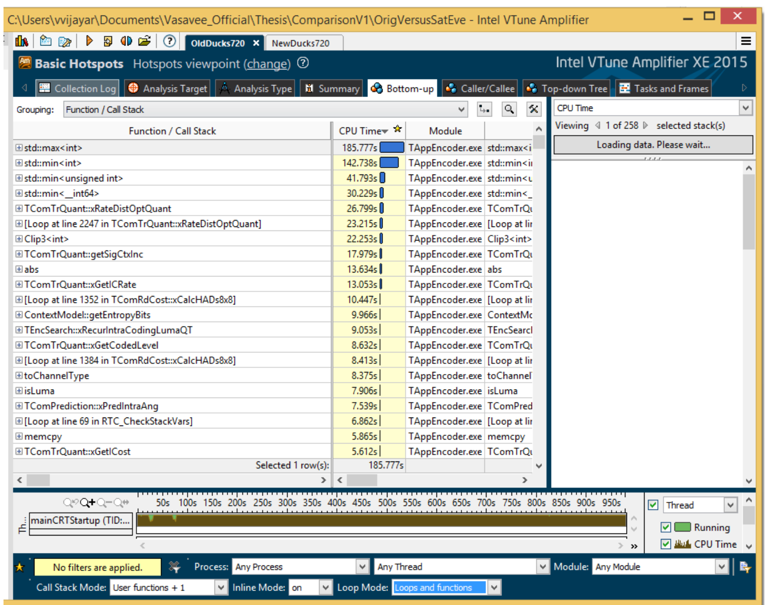


Figure 7-30 Hotspot analysis bottom-up for DucksTakeOff (Original HM)

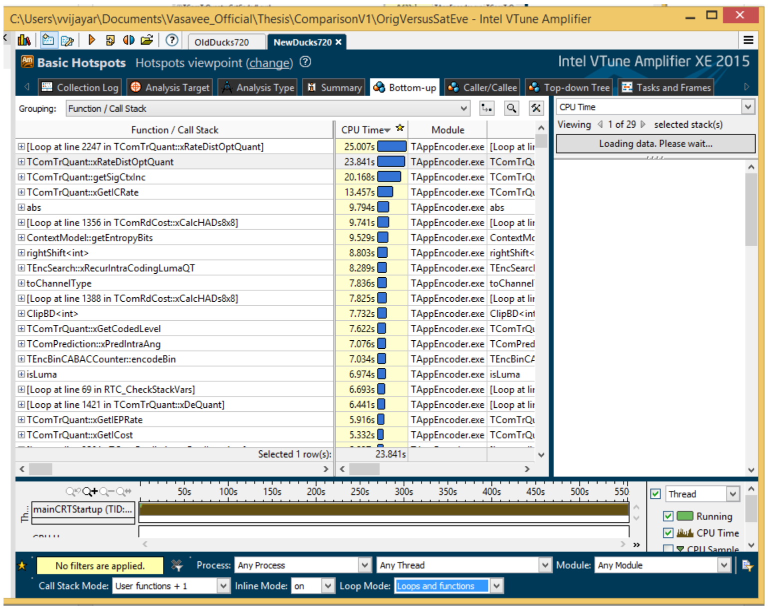


Figure 7-31 Hotspot analysis bottom-up for DucksTakeOff (Optimized HM)

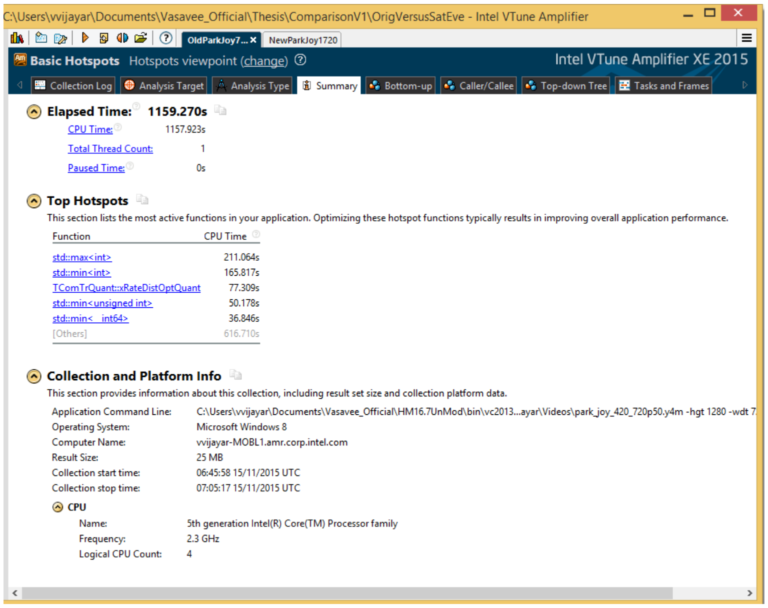


Figure 7-32 Hotspot analysis summary for ParkJoy (Original HM)

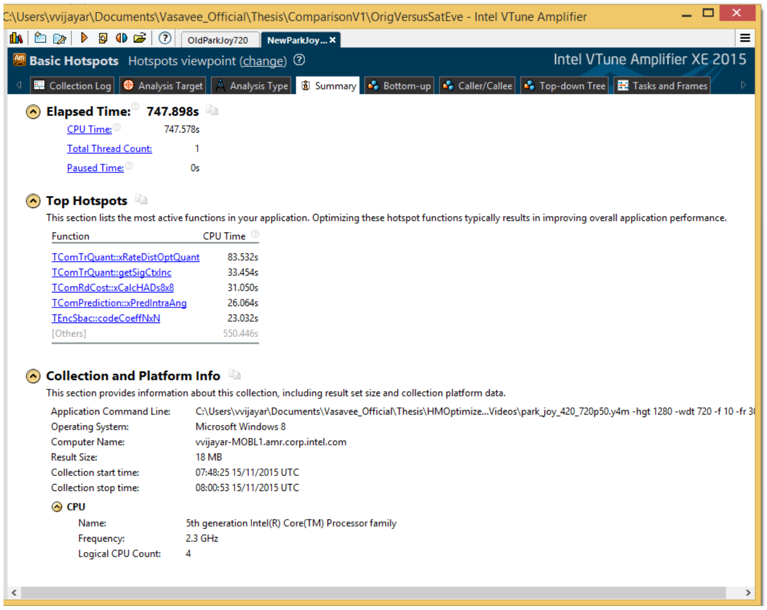


Figure 7-33 Hotspot analysis summary for ParkJoy (Optimized HM)

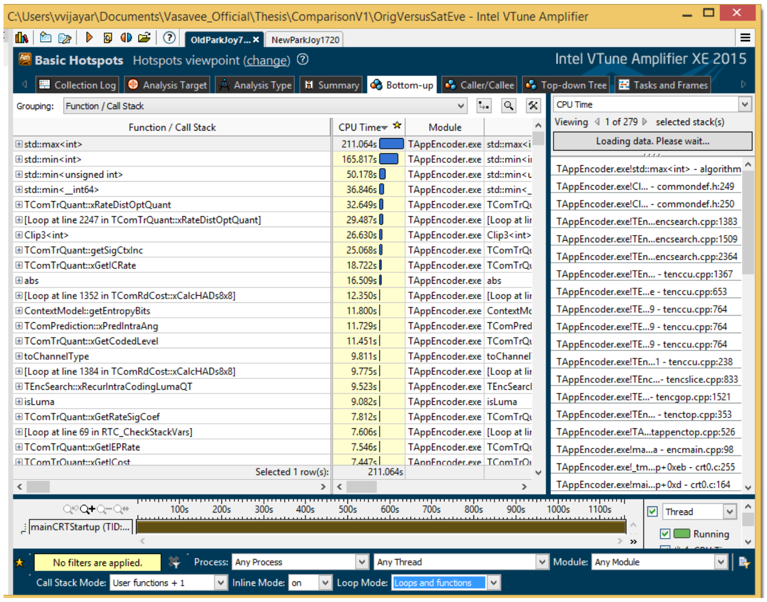


Figure 7-34 Hotspot analysis bottom-up for ParkJoy (Original HM)

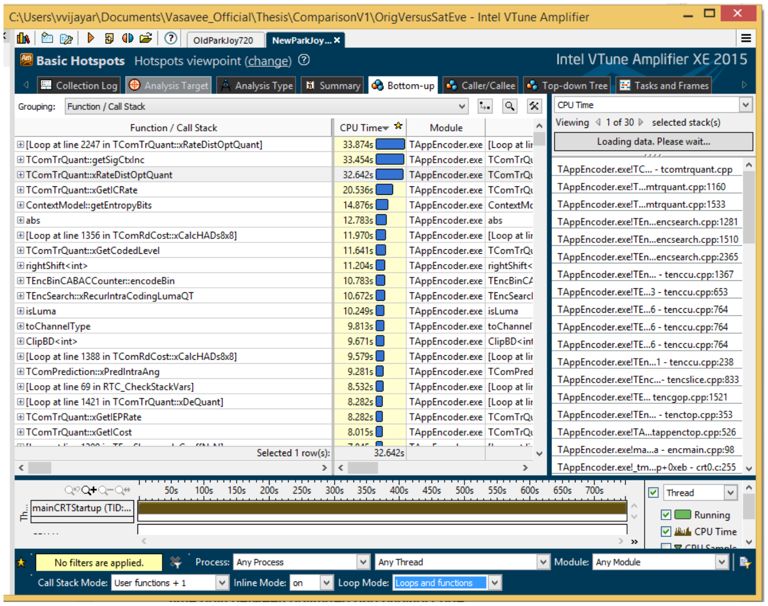


Figure 7-35 Hotspot analysis bottom-up for ParkJoy (Optimized HM)

7.2.2 Time gain between optimized and original code:

CrowdRun720p: Negative difference in elapsed time (New-Old) shows the reduction in encoding time and hotspot removal

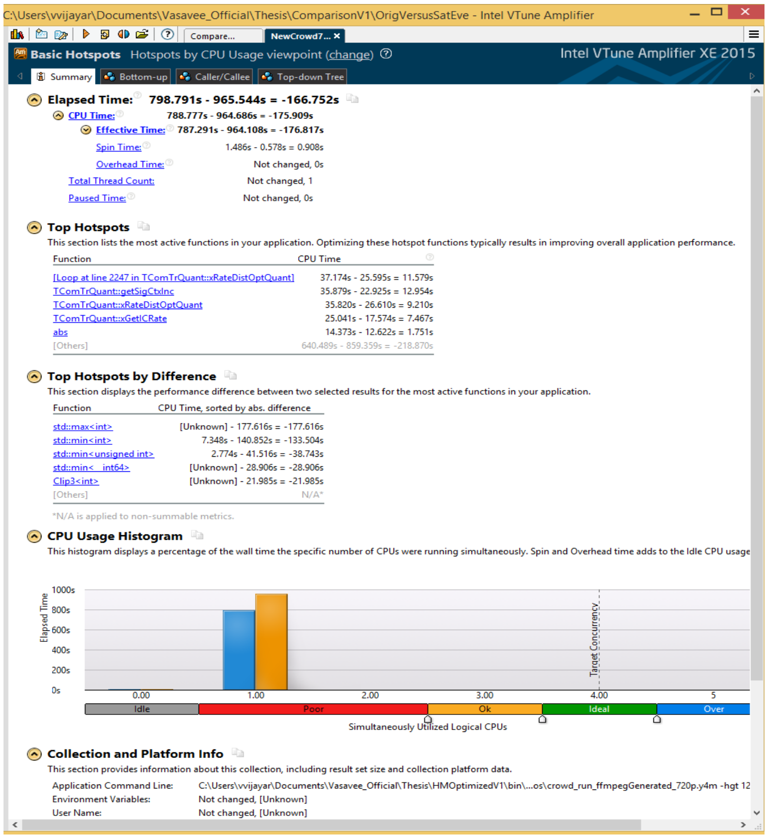


Figure 7-36 Crowdrun 720p difference in vTune encoding time

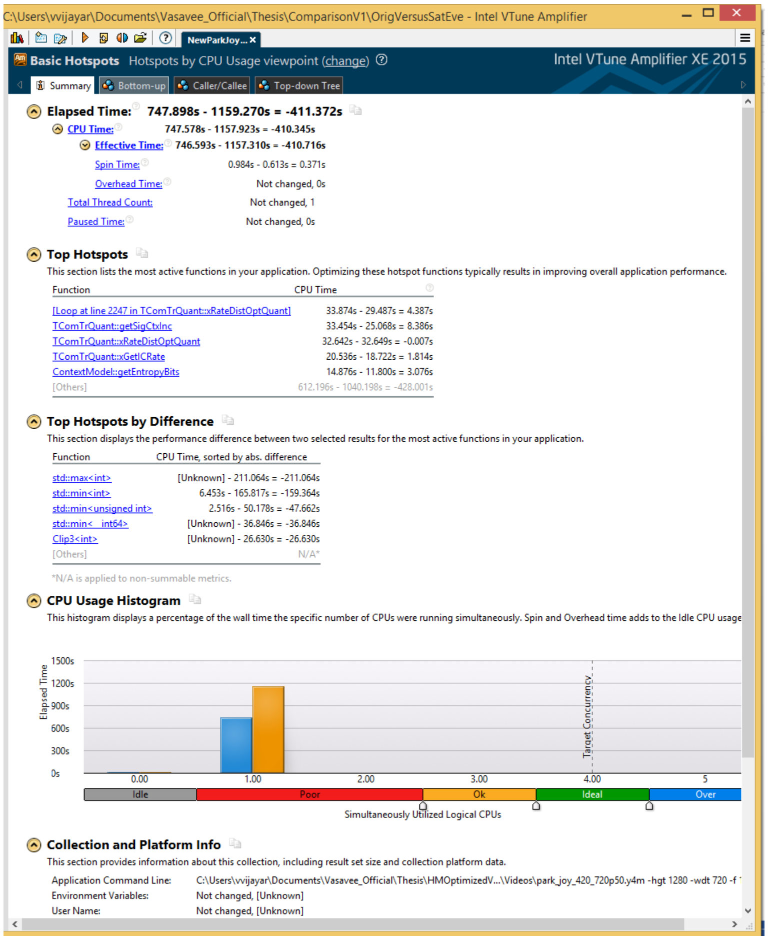


Figure 7-37 Parkjoy 720p difference in vTune encoding time

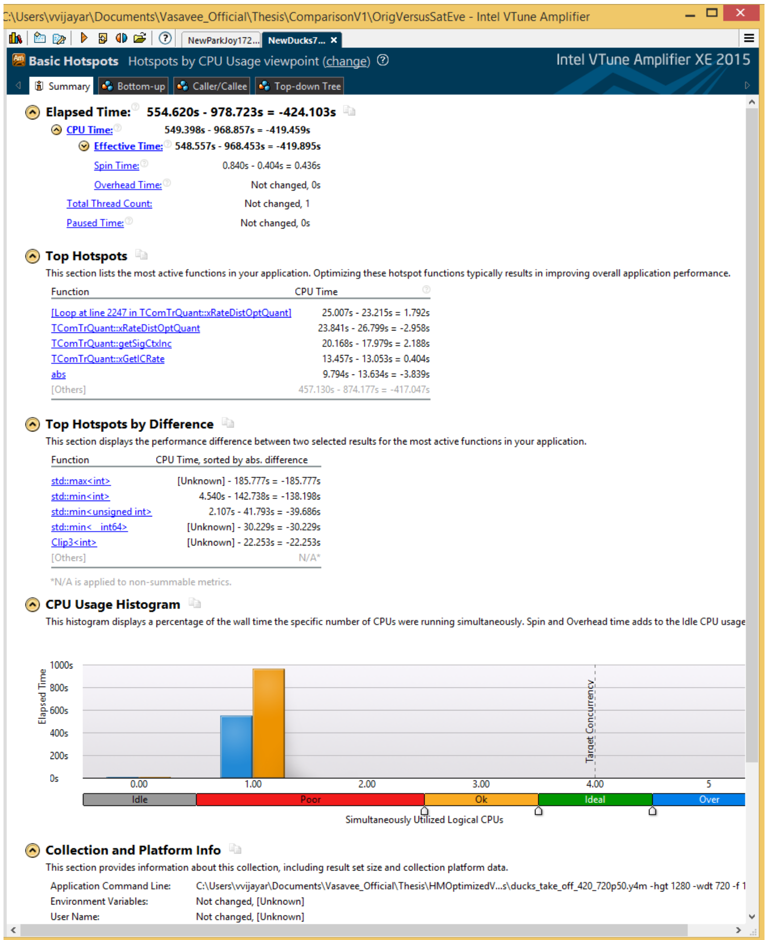


Figure 7-38 DucksTakeOff720p difference in vTune encoding time

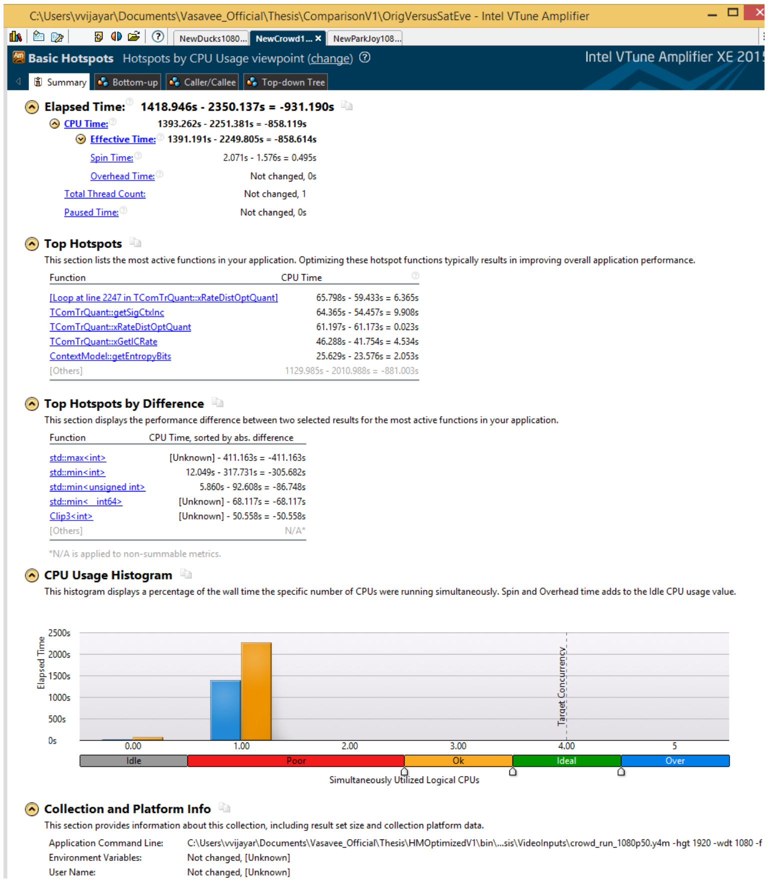


Figure 7-39 CrowdRun1080p difference in vTune encoding time

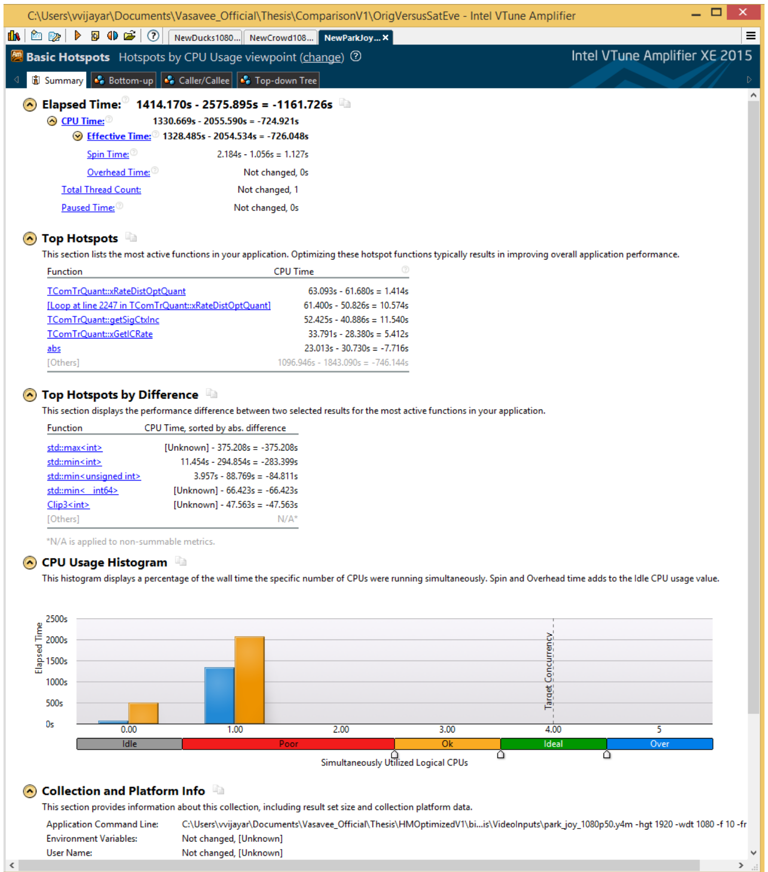


Figure 7-40 ParkJoy1080p difference in vTune encoding time

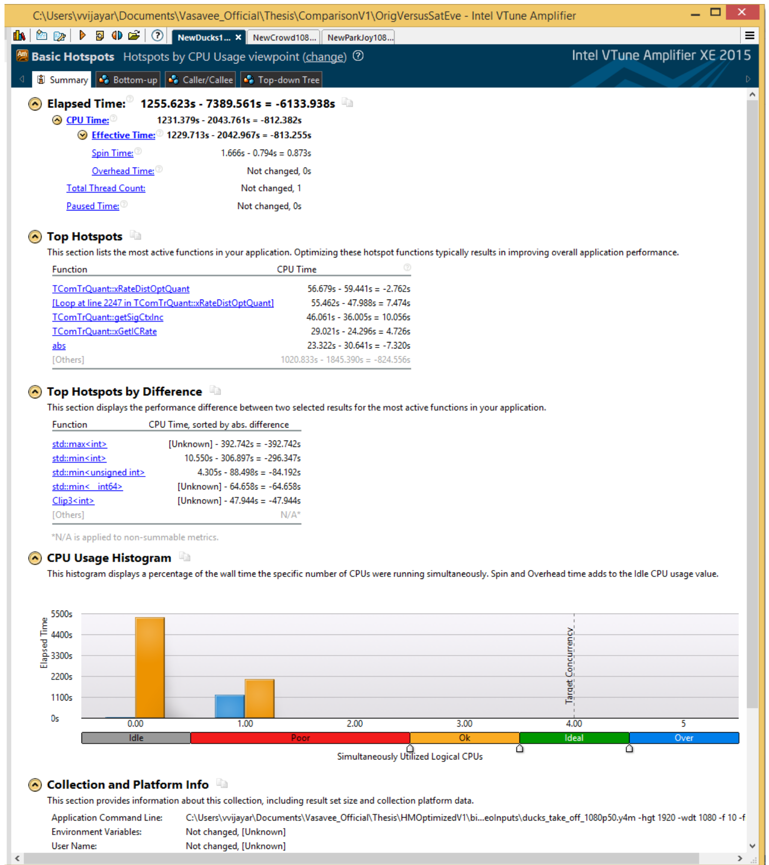


Figure 7-41 DucksTakeOff1080p difference in vTune encoding time

Summary of vTune analysis: (Encoding in Debug mode – No. of frames=10; Frame rate= 30; Encoder Profile=Intra_main)

Table 7-1 Summary of Intel® vTune™ Analysis

Name of video sequence	Encoding time of original HM16.7	Encoding time after optimization	Time gain	Functions/Loops optimized
CrowdRun_1080p	2251.381s	1393.262s	858.119s	1. Clip3<> 2. ClipBD<> 3. For loops throughout the code 4. QPParam::QPParam
CrowdRun_720p	964.686s	788.777s	175.909s	
DucksTakeOff_1080p	2043.761s	1231.379s	812.382s	
DucksTakeOff_720p	968.857s	549.398s	419.459s	
ParkJoy_1080p	2055.590s	1330.669s	724.921s	
ParkJoy_720p	1157.923s	747.578s	410.345s	

Inference:

Analysis of hotspots in vTune amplifier show that the hotspots produced by all the sequences are common and hence they are all targeted and optimized using OpenMP for loops. A snapshot of common functions/loops for one of the hotspots is as shown below:

Function	CPU Time
std::max<int>	411.163s
std::min<int>	317.731s
TComTrQuant::xRateDistOptQuant	152.040s
std::min<unsigned int>	92.608s
std::min<__int64>	68.117s
[Others]	1209.722s

Figure 7-42 Common hotspots before optimization

Function	CPU Time
TComTrQuant::xRateDistOptQuant	60.571s
TComRdCost::xCalcHADs8x8	24.988s
TComTrQuant::getSigCtxInc	20.168s
TComPrediction::xPredIntraAng	19.705s
TEncSbac::codeCoeffNxN	14.745s
[Others]	409.221s

Figure 7-43 Common hotspots after optimization

std::max<int>	411.163s	TAppEncoder.exe	std::max<int>(int const
Clip3<int>	411.153s	TAppEncoder.exe	Clip3<int>(int,int,int)
[Loop at line 1421 in TComTrQuant::xDeQuant]← TComTrQuant::x	163.915s	TAppEncoder.exe	[Loop at line 1421 in TCo
ClipBD<int>	95.321s	TAppEncoder.exe	ClipBD<int>(int,int)
[Loop at line 577 in partialButterflyInverse8]← [Loop at line 558 in p	38.261s	TAppEncoder.exe	[Loop at line 577 in parti
[Loop at line 453 in fastInverseDst]← [Loop at line 445 in fastInverse	33.132s	TAppEncoder.exe	[Loop at line 453 in fastIı
[Loop at line 693 in partialButterflyInverse16]← [Loop at line 662 in	31.754s	TAppEncoder.exe	[Loop at line 693 in parti
[Loop at line 840 in partialButterflyInverse32]← [Loop at line 794 in	28.670s	TAppEncoder.exe	[Loop at line 840 in parti
[Loop at line 482 in partialButterflyInverse4]← partialButterflyInvers	12.274s	TAppEncoder.exe	[Loop at line 482 in parti
[Loop at line 336 in TComPrediction::xPredIntraAng]← TComPredi	2.878s	TAppEncoder.exe	[Loop at line 336 in TCor
[Loop at line 355 in TComSampleAdaptiveOffset::offsetBlock]← [Lc	1.873s	TAppEncoder.exe	[Loop at line 355 in TCor
QpParam::QpParam← QpParam::QpParam← TEncSearch::xIntraCo	0.978s	TAppEncoder.exe	QpParam::QpParam(int,
getScaledChromaQP← QpParam::QpParam← QpParam::QpParam	0.733s	TAppEncoder.exe	getScaledChromaQP
[Loop at line 536 in TComSampleAdaptiveOffset::offsetBlock]← [Lc	0.409s	TAppEncoder.exe	[Loop at line 536 in TCor
TComLoopFilter::xPelFilterLuma← [Loop at line 665 in TComLoopF	0.379s	TAppEncoder.exe	TComLoopFilter::xPelFilt
[Loop at line 602 in TComLoopFilter::xEdgeFilterLuma]← TComLoc	0.219s	TAppEncoder.exe	[Loop at line 602 in TCor
TComLoopFilter::xPelFilterChroma← [Loop at line 807 in TComLoo	0.158s	TAppEncoder.exe	TComLoopFilter::xPelFilt
[Loop at line 782 in TComLoopFilter::xEdgeFilterChroma]← [Loop a	0.060s	TAppEncoder.exe	[Loop at line 782 in TCor
getScaledChromaQP← [Loop at line 782 in TComLoopFilter::xEdge	0.060s	TAppEncoder.exe	getScaledChromaQP
[Loop at line 457 in TEncSampleAdaptiveOffset::deriveOffsets]← TE	0.020s	TAppEncoder.exe	[Loop at line 457 in TEnc
[Loop at line 507 in TComSampleAdaptiveOffset::offsetBlock]← [Lc	0.020s	TAppEncoder.exe	[Loop at line 507 in TCor
TEncCu::xComputeQP← TEncCu::xCompressCU← [Loop at line 744	0.020s	TAppEncoder.exe	TEncCu::xComputeQP(c
[Loop at line 444 in TComSampleAdaptiveOffset::offsetBlock]← [Lc	0.019s	TAppEncoder.exe	[Loop at line 444 in TCor
ContextModel::init← [Loop at line 72 in ContextModel3DBuffer::initB	0.010s	TAppEncoder.exe	ContextModel::init(int,in
std::min<int>	317.731s	TAppEncoder.exe	std::min<int>(int const
std::min<unsigned int>	92.608s	TAppEncoder.exe	std::min<unsigned int>(
std::min<__int64>	68.117s	TAppEncoder.exe	std::min<__int64>(__int6

Figure 7-44 Function hotpots in HM16.7 for all video sequences used

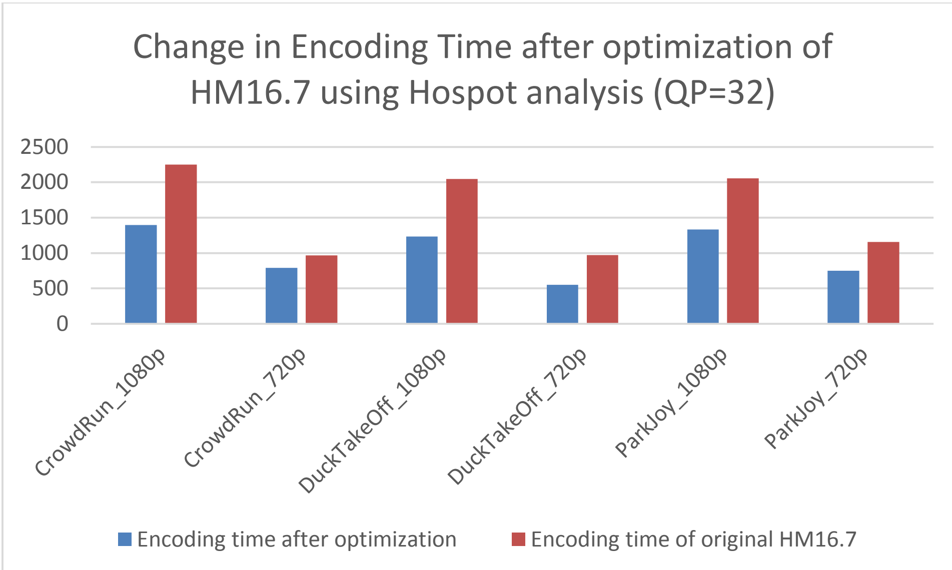


Figure 7-45 Change in Encoding Time before and after Intel ® vTune™ analysis

Optimal configuration settings adopted for best encoding time in encoder_intra_main.cfg:

#======== File I/O =====================

BitstreamFile : str.bin

ReconFile : rec.yuv

PrintSequenceMSE : 1

#======== Profile ================

Profile : main

#======== Unit definition ================

MaxCUWidth : 64 # Maximum coding unit width in pixel

MaxCUHeight : 64 # Maximum coding unit height in pixel

MaxPartitionDepth : 4 # Maximum coding unit depth

QuadtreeTULog2MaxSize : 5 # Log2 of maximum transform size for

```
                                      # quadtree-based TU coding (2...6)
QuadtreeTULog2MinSize        : 2          # Log2 of minimum transform size for
                                      # quadtree-based TU coding (2...6)
QuadtreeTUMaxDepthInter      : 3
QuadtreeTUMaxDepthIntra      : 3

#======== Coding Structure =============
IntraPeriod                  : 1          # Period of I-Frame ( -1 = only first)
DecodingRefreshType          : 0          # Random Accesss 0:none, 1:CRA, 2:IDR,
3:Recovery Point SEI
GOPSize                      : 1          # GOP Size (number of B slice = GOPSize-1)
#       Type POC QPoffset QPfactor tcOffsetDiv2 betaOffsetDiv2  temporal_id
#ref_pics_active #ref_pics reference pictures

#=========== Motion Search =============
FastSearch                   : 1          # 0:Full search  1:TZ search
SearchRange                  : 64         # (0: Search range is a Full frame)
HadamardME                   : 1          # Use of hadamard measure for fractional ME
FEN                          : 1          # Fast encoder decision
FDM                          : 1          # Fast Decision for Merge RD cost

#======== Quantization =============
QP                           : 32         # Quantization parameter(0-51)
MaxDeltaQP                   : 0          # CU-based multi-QP optimization
```

MaxCuDQPDepth : 0 # Max depth of a minimum CuDQP for sub-LCU-level delta QP

DeltaQpRD : 0 # Slice-based multi-QP optimization

RDOQ : 1 # RDOQ

RDOQTS : 1 # RDOQ for transform skip

#=========== Deblock Filter ============

LoopFilterOffsetInPPS : 1 # Dbl params: 0=varying params in SliceHeader, param = base_param + GOP_offset_param; 1 (default) =constant params in PPS, param = base_param)

LoopFilterDisable : 0 # Disable deblocking filter (0=Filter, 1=No Filter)

LoopFilterBetaOffset_div2 : 0 # base_param: -6 ~ 6

LoopFilterTcOffset_div2 : 0 # base_param: -6 ~ 6

DeblockingFilterMetric : 0 # blockiness metric (automatically configures deblocking parameters in bitstream). Applies slice-level loop filter offsets (LoopFilterOffsetInPPS and LoopFilterDisable must be 0)

#=========== Misc. ============

InternalBitDepth : 8 # codec operating bit-depth

#=========== Coding Tools =================

SAO : 1 # Sample adaptive offset (0: OFF, 1: ON)

AMP : 1 # Asymmetric motion partitions (0: OFF, 1: ON)

TransformSkip : 1 # Transform skipping (0: OFF, 1: ON)

TransformSkipFast : 1 # Fast Transform skipping (0: OFF, 1: ON)

SAOLcuBoundary : 0 # SAOLcuBoundary using non-deblocked pixels (0: OFF, 1: ON)

#=========== Slices ===============

SliceMode : 0 # 0: Disable all slice options.

1: Enforce maximum number of LCU in an slice,

2: Enforce maximum number of bytes in an 'slice'

3: Enforce maximum number of tiles in a slice

SliceArgument : 1500 # Argument for 'SliceMode'.

If SliceMode==1 it represents max. SliceGranularity-sized blocks per slice.

If SliceMode==2 it represents max. bytes per slice.

If SliceMode==3 it represents max. tiles per slice.

LFCrossSliceBoundaryFlag : 1 # In-loop filtering, including ALF and DB, is across or not across slice boundary.

0:not across, 1: across

#=========== PCM ===============

PCMEnabledFlag : 0 # 0: No PCM mode

PCMLog2MaxSize : 5 # Log2 of maximum PCM block size.

PCMLog2MinSize : 3 # Log2 of minimum PCM block size.

PCMInputBitDepthFlag : 1 # 0: PCM bit-depth is internal bit-depth. 1: PCM bit-depth is input bit-depth.

PCMFilterDisableFlag : 0 # 0: Enable loop filtering on I_PCM samples.

1: Disable loop filtering on I_PCM samples.

#============ Tiles ================

TileUniformSpacing : 0 # 0: the column boundaries are indicated by

TileColumnWidth array, the row boundaries are indicated by TileRowHeight array

1: the column and row boundaries are distributed

uniformly

NumTileColumnsMinus1 : 0 # Number of tile columns in a picture

minus 1

TileColumnWidthArray : 2 3 # Array containing tile column width values

in units of CTU (from left to right in picture)

NumTileRowsMinus1 : 0 # Number of tile rows in a picture minus 1

TileRowHeightArray : 2 # Array containing tile row height values in

units of CTU (from top to bottom in picture)

LFCrossTileBoundaryFlag : 1 # In-loop filtering is across or not across

tile boundary.

0:not across, 1: across

#============ WaveFront ================

WaveFrontSynchro : 0 # 0: No WaveFront synchronisation

(WaveFrontSubstreams must be 1 in this case).

>0: WaveFront synchronises with the LCU above

and to the right by this many LCUs.

#=========== Quantization Matrix =================

ScalingList : 0 # ScalingList 0 : off, 1 : default, 2 : file read

ScalingListFile : scaling_list.txt # Scaling List file name. If file is not exist, use Default Matrix.

#============ Lossless ================

TransquantBypassEnableFlag : 0 # Value of PPS flag.

CUTransquantBypassFlagForce: 0 # Force transquant bypass mode, when transquant_bypass_enable_flag is enabled

DO NOT ADD ANYTHING BELOW THIS LINE

DO NOT DELETE THE EMPTY LINE BELOW

Final Encoder Performance Comparison between Original and Optimized code after Parallelization using OpenMP

Encoder versions: HM16.7_Original.exe and HM16.7_Modified.exe

Configuration Files: encoder_intra_main.cfg

VideoSequences: DucksTakeOff_1080p.y4m, CrowdRun_1080p.y4m, ParkJoy_1080p.y4m, DucksTakeOff_720p.y4m, CrowdRun_720p.y4m, ParkJoy_720p.y4m

Quantization Parameters: 22, 24, 26, 28, 30, 32

Metrics Generated by encoder: Encoding time and PSNR

Metrics Generated from Matlab: BD-rate and BD-PSNR

The following PSNR, Encoding time and RD plots are described for 6 different Quantization Parameters (QPs) – 22,24,26,28,30,32 and for two different resolutions (1080p and 720p) for three different sequences(DucksTakeOff (easy), ParkJoy(medium) and CrowdRun(heavy)) classified in terms of motion in each of the video. These are chosen keeping in mind that HEVC is designed for high resolution videos and that the optimized encoder is tested for low to high quality and low to high complexity.

7.2.3 Comparison between original and optimized HM16.7 for CrowdRun, ParkJoy and DucksTakeOff.y4m

Table 7-2 Unoptimized versus Optimized PSNR, Bitrate and Encoding Time Comparison for CrowdRun.y4m

CrowdRun 1080p							
Unoptimized					Optimized		
QP	Unop_PSNR (dB)	Unop_Bitrate (kbps)	Unop_Encoding Time (sec)		Op_PSNR (dB)	Op_Bitrate (kbps)	Op_Encoding Time (sec)
22	41.1217	121080.5	1682.644		42.9996	124712	1141.764
24	39.634	97863.02	1580.749		41.4933	100798	1076.698
26	38.1949	78592.27	1511.639		39.8935	80949	1010.887
28	36.8895	63248.02	1445.685		38.582	65145	955.791
30	35.6993	51549.24	1420.278		37.4187	53095	926.984
32	34.4019	40339.92	1351.443		36.1523	41549	889.656
CrowdRun 720p							
Unoptimized					Optimized		
QP	Unop_PSNR (dB)	Unop_Bitrate (kbps)	Unop_Encoding Time (sec)		Op_PSNR (dB)	Op_Bitrate (kbps)	Op_Encoding Time (sec)
22	41.9984	52294.08	852.708		43.6744	54385	600.184
24	40.469	43645.56	870.224		42.2154	45390	570.931
26	38.8825	35710.3	842.953		40.6277	37138.71	551.382
28	37.4206	29101.03	803.372		39.2501	30265.07	466.538
30	36.0916	23893.51	691.588		37.9896	24849.25	446.114
32	34.6634	18875.47	701.46		36.5977	19630.49	461.282

Figure 7-46 Crowd Run (1920x1080 and 1280x720)

Time

Table 7-3 Unoptimized versus Optimized PSNR, Bitrate and Encoding Time Comparison

for DucksTakeOff.y4m

DucksTakeOff 1080p						
Unoptimized				Optimized		
QP	Unop_PSNR (dB)	Unop_Bitrate (kbps)	Unop_Encoding Time (sec)	Op_PSNR (dB)	Op_Bitrate (kbps)	Op_Encoding Time (sec)
22	39.778	100808.2	1774.56	42.778	103529	1102.77
24	38.8434	81808.18	1674.167	41.2291	84098	1077.34
26	37.2445	55678.63	1521.576	39.5272	57348	1021.542
28	36.1032	38640.07	1434.559	38.2288	39799	952.338
30	35.402	30438.7	1400.911	37.1078	31351	901.13
32	34.6431	23723.59	1332.25	36.067	24434	842.734
DucksTakeOff 720p						
Unoptimized				Optimized		
QP	Unop_PSNR (dB)	Unop_Bitrate (kbps)	Unop_Encoding Time (sec)	Op_PSNR (dB)	Op_Bitrate (kbps)	Op_Encoding Time (sec)
22	40.8364	47308.42	825.291	43.1076	49200	618.142
24	39.3717	36513.14	874.066	41.5889	37973.67	584.044
26	38.1005	28645.13	836.714	39.9948	29790.93	543.708
28	36.9992	22931.76	728.015	38.7573	23849.03	519.276
30	36.0325	18933.34	788.087	37.6741	19690.67	437.925
32	34.8912	14973.36	745.495	36.5405	15572.29	429.66

Figure 7-47 DucksTakeOff (1920x1080 and 1280x720)

Time

Table 7-4 Unoptimized versus Optimized PSNR, Bitrate and Encoding Time Comparison for ParkJoy.y4m

ParkJoy 1080p						
	Unoptimized			Optimized		
QP	Unop_PSNR (dB)	Unop_Bitrate (kbps)	Unop_Encoding Time (sec)	Op_PSNR (dB)	Op_Bitrate (kbps)	Op_Encoding Time (sec)
22	41.955	72435	1465.99	43.542	74435	1045
24	40.9546	64626.36	1355.548	42.5243	66410	932.588
26	39.6752	52340.5	1293.34	41.1383	53386	858.698
28	38.4539	42236.5	1242.713	40.0091	43107	820.852
30	37.3231	34502.64	1222.712	38.9811	35192	814.035
32	36.039	26827.25	1174.481	37.7777	27263	775.656
ParkJoy 720p						
	Unoptimized			Optimized		
QP	Unop_PSNR (dB)	Unop_Bitrate (kbps)	Unop_Encoding Time (sec)	Op_PSNR (dB)	Op_Bitrate (kbps)	Op_Encoding Time (sec)
22	42.7536	41998.27	801.541	44.1995	43677	480.617
24	41.3585	35451.74	769.664	42.9016	36869.81	458.996
26	39.8356	29318.83	662.207	41.4162	30491.59	487.858
28	38.3981	24088.75	712.488	40.1136	25052.3	427.628
30	37.0607	19888.2	615.459	38.8843	20683.73	421.513
32	35.5803	15643.85	590.02	37.4782	16269.6	444.31

Figure 7-48 ParkJoy (1920x1080 and 1280x720)

7.2.4 PSNR comparison plots between un-optimized and optimized versions of HM16.7 (HEVC):

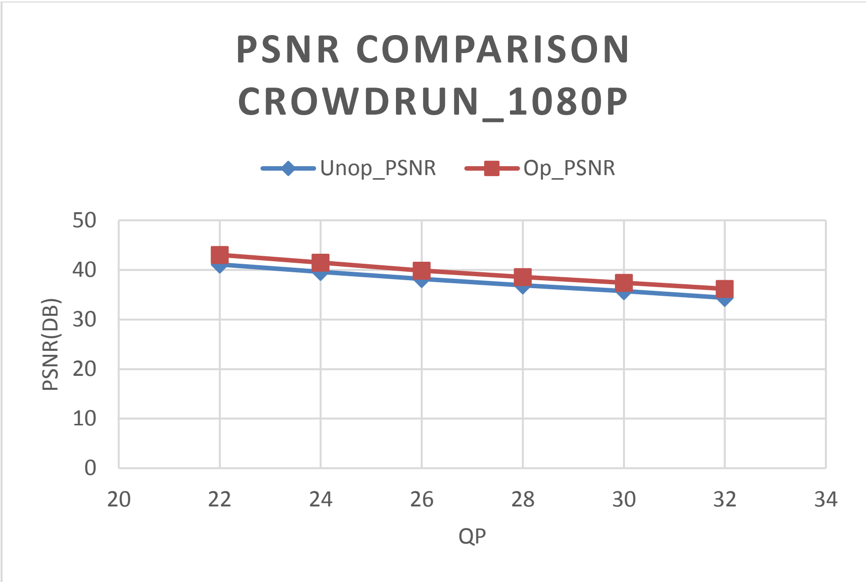


Figure 7-49 PSNR comparison plot for CrowdRun_1080p.y4m

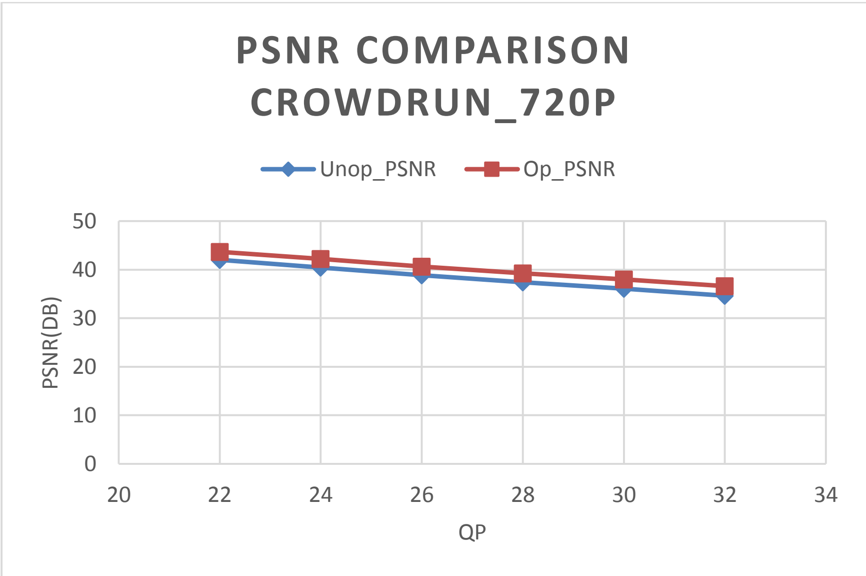


Figure 7-50 PSNR comparison plot for CrowdRun_720p.y4m

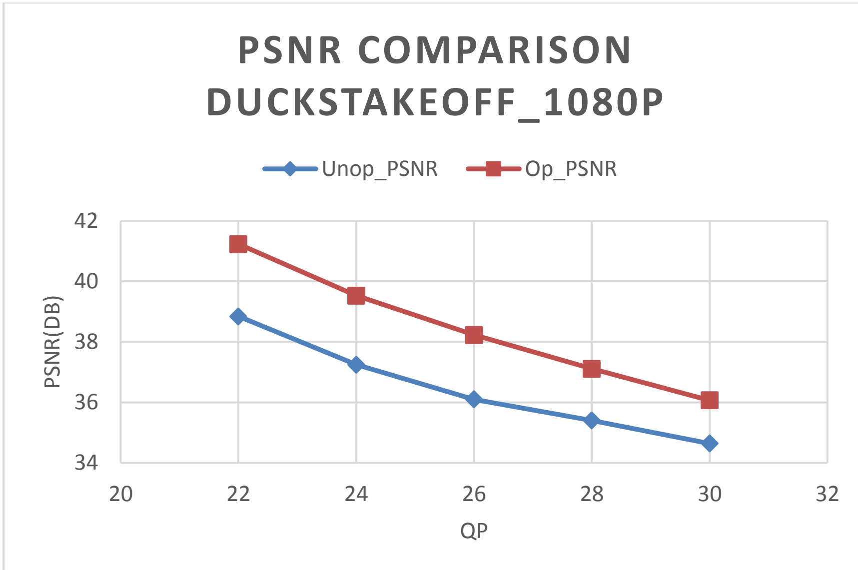


Figure 7-51 : PSNR comparison plot for DucksTakeOff_1080p.y4m

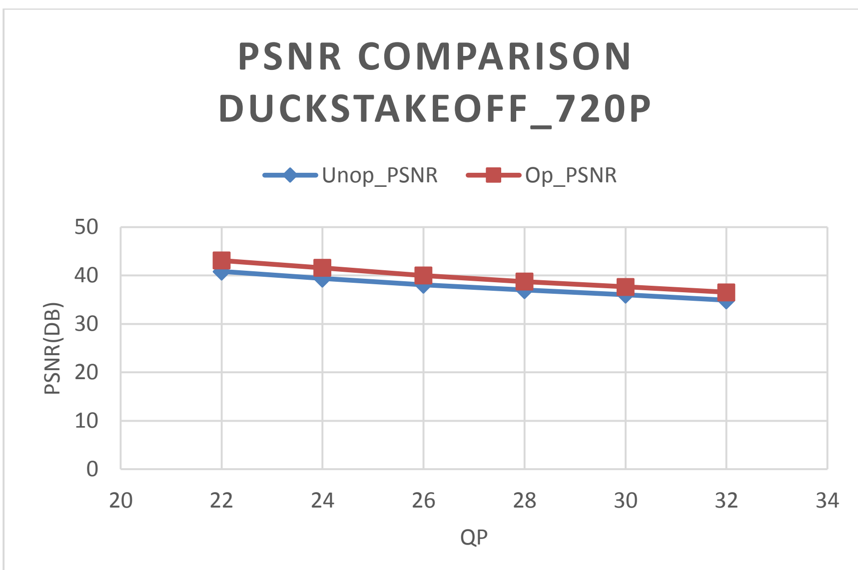


Figure 7-52 PSNR comparison plot for DucksTakeOff_720p.y4m

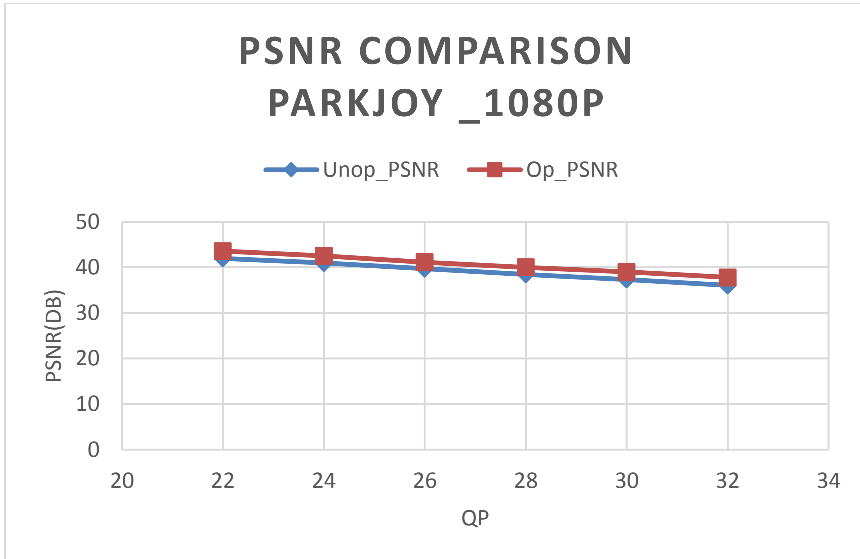


Figure 7-53 PSNR comparison plot for ParkJoy_1080p.y4m

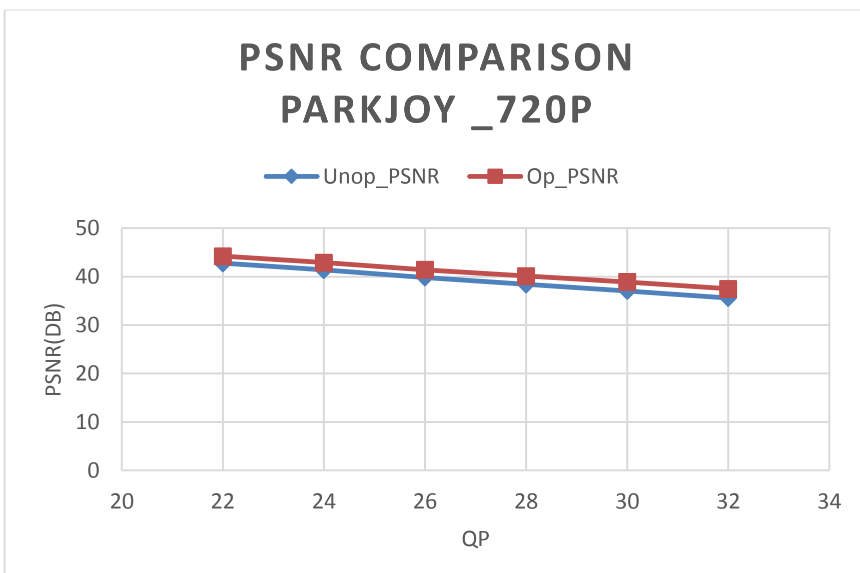


Figure 7-54 PSNR comparison plot for ParkJoy_720p.y4m

Figures 8-2 to 8-7 illustrate the difference in PSNR between the original HM software encoder and the optimized HM software encoder. These plots show that the optimized software has a slight increase in PSNR for every QP and for each of the three sequences, thus ensuring that the quality of the video is not degraded.

7.2.5 Encoding Time comparison plots between un-optimized and optimized versions of HM16.7 (HEVC):

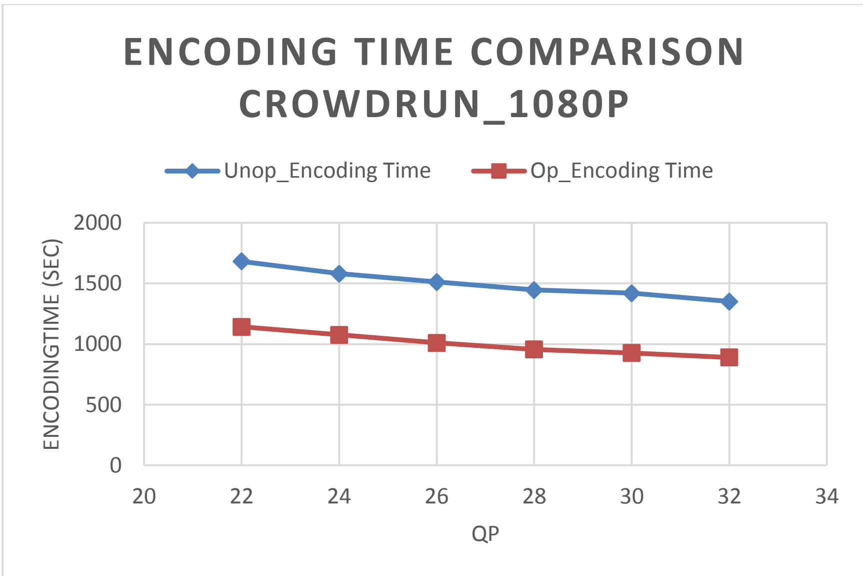


Figure 7-55 Encoding Time Comparison plot for CrowdRun_1080p.y4m

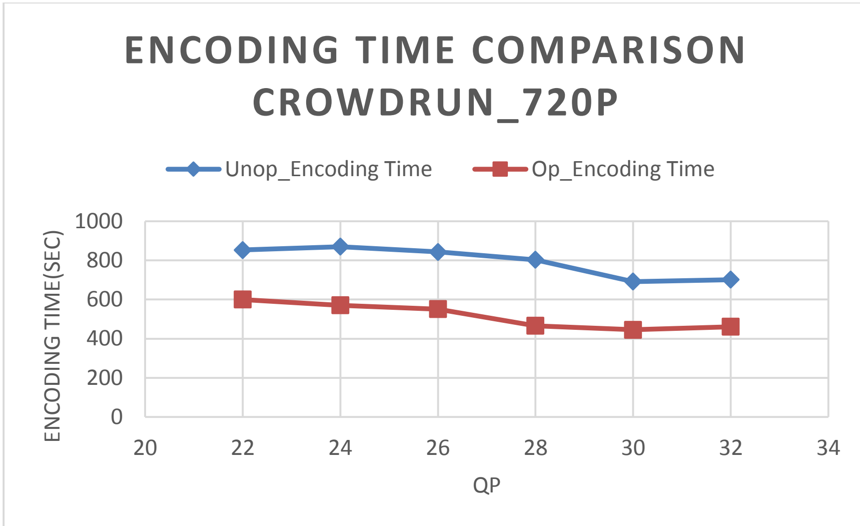


Figure 7-56 Encoding Time Comparison plot for CrowdRun_720p.y4m

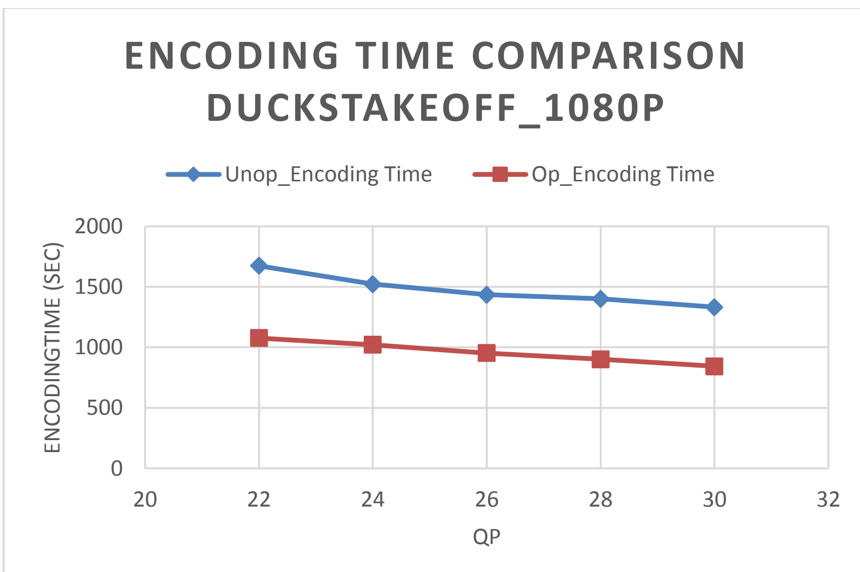


Figure 7-57 Encoding Time Comparison plot for DucksTakeOff_1080p.y4m

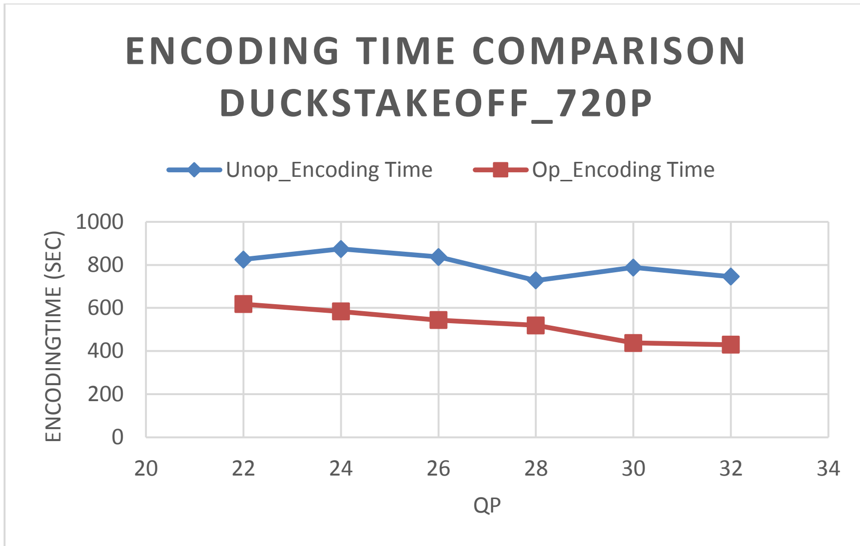


Figure 7-58 Encoding Time Comparison plot for DucksTakeOff_720p.y4m

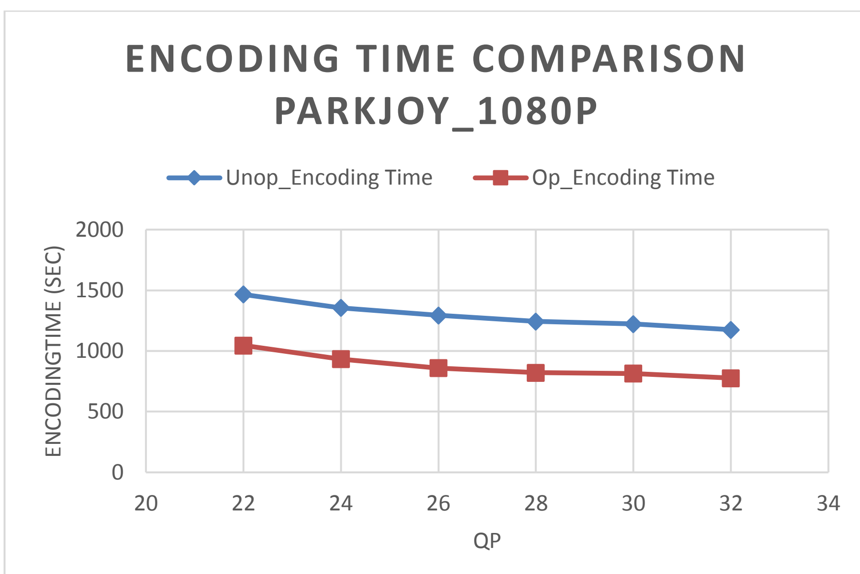


Figure 7-59 Encoding Time Comparison plot for ParkJoy_1080p.y4m

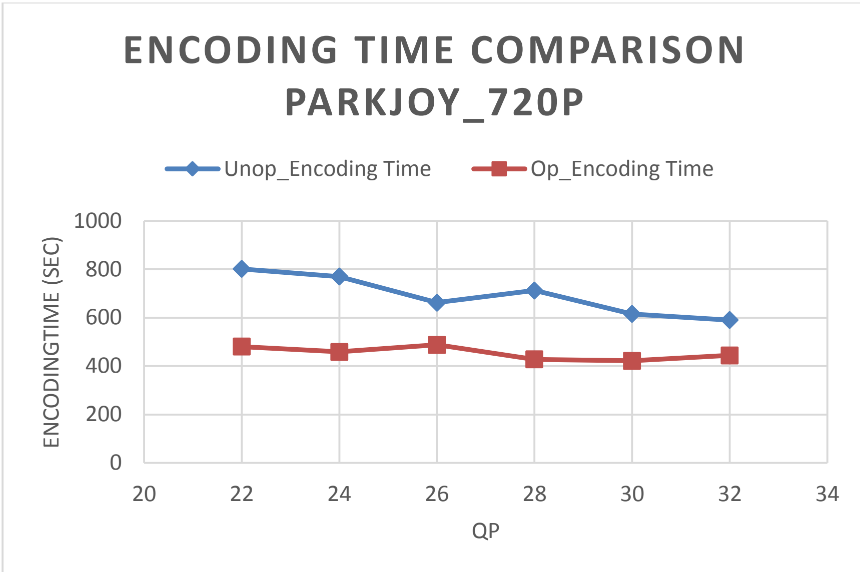


Figure 7-60 Encoding Time Comparison plot for ParkJoy_720p.y4m

Figures 8-8 to 8-13 show the difference in Encoding time between the original HM software encoder and the optimized HM software encoder. It can be observed from these plots that the purpose of this thesis is successfully accomplished i.e., reduction in encoding time with no loss of quality using parallel programming with OpenMP.

7.2.6 RD-plot comparison plots between un-optimized and optimized versions of HM16.7 (HEVC):

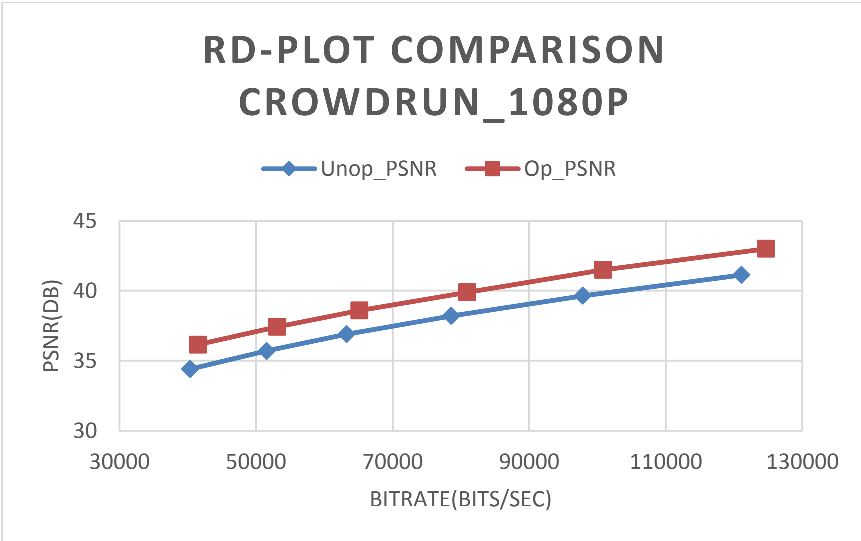


Figure 7-61 RD-plot comparison for CrowdRun_1080p.y4m

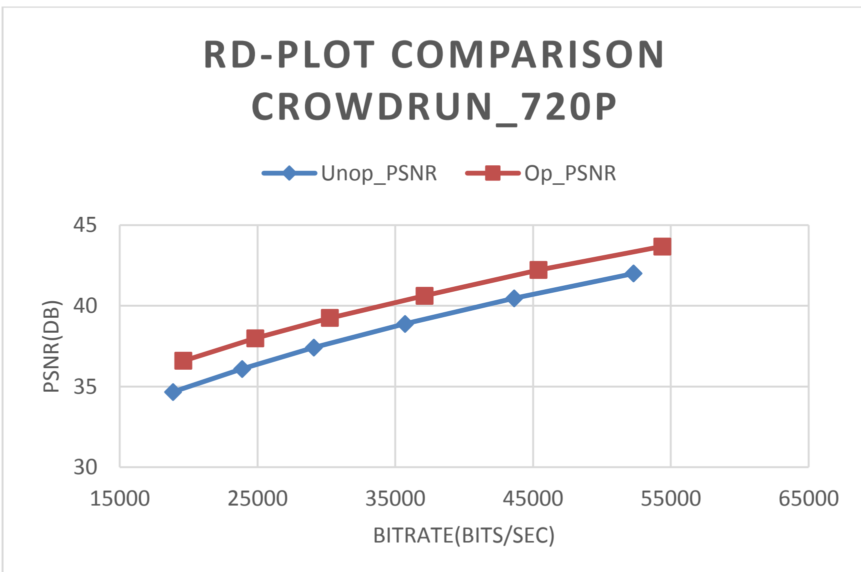


Figure 7-62 RD-plot comparison for CrowdRun_720p.y4m

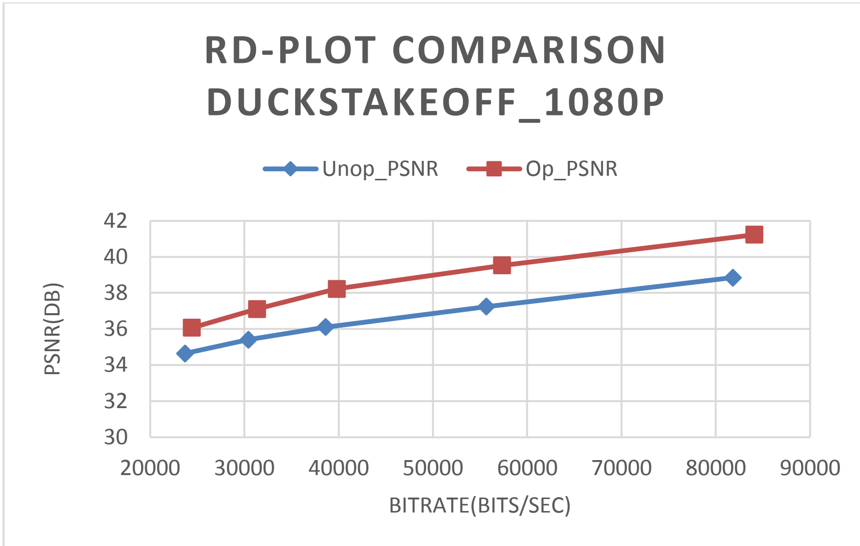


Figure 7-63 RD-plot comparison for DuckstakeOff_1080p.y4m

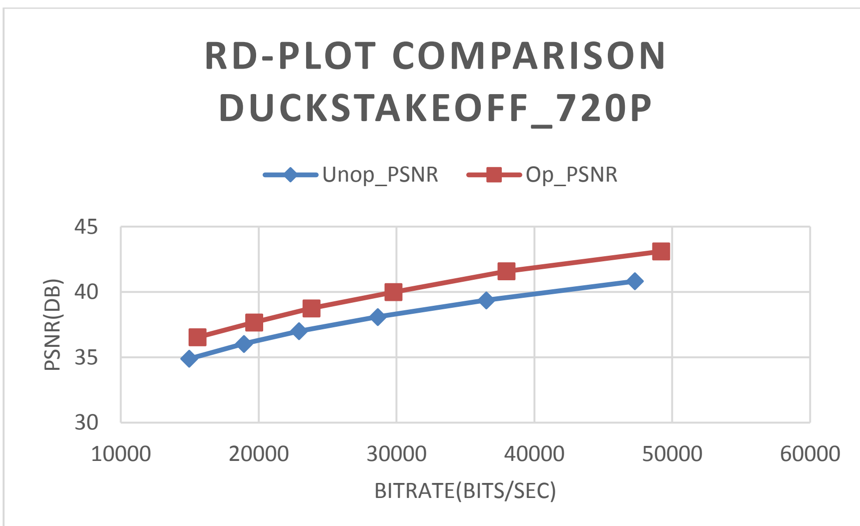


Figure 7-64 RD-plot comparison for DuckstakeOff_720p.y4m

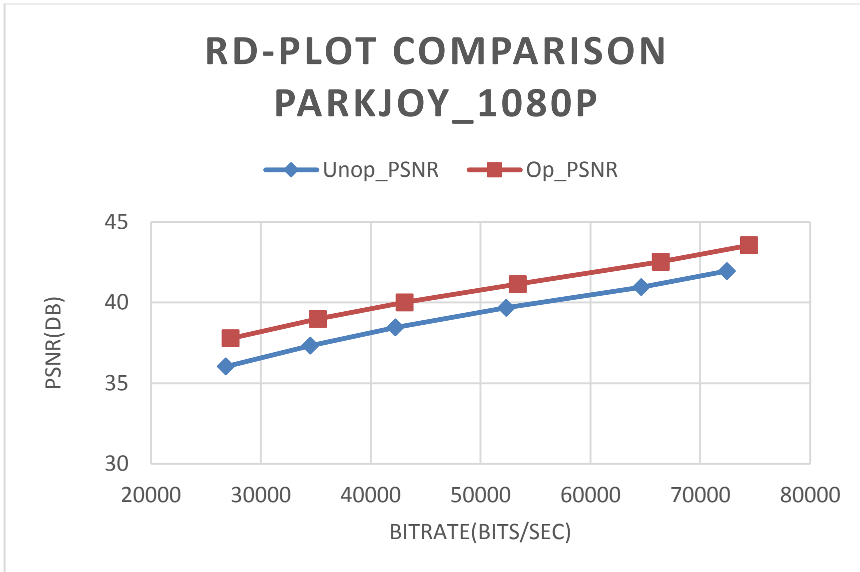


Figure 7-65 RD-plot comparison for ParkJoy_1080p.y4m

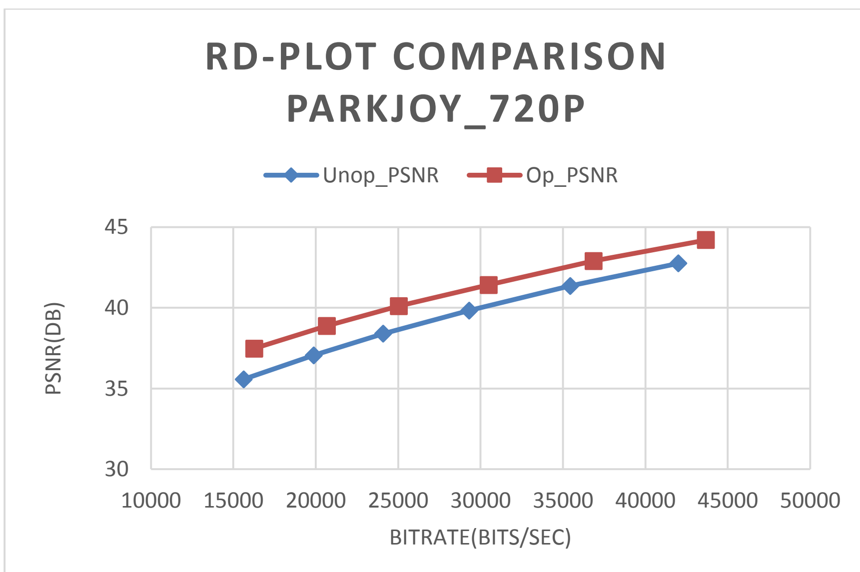


Figure 7-66 RD-plot comparison for ParkJoy_720p.y4m

Figures 8-14 to 8-19 show the RD plot comparison between the original and optimized HM encoders. It is very evident from these plots that a slight bitrate increase has been encountered with the optimized software with no loss of quality (PSNR).

Chapter 8

CONCLUSIONS AND FUTURE WORK

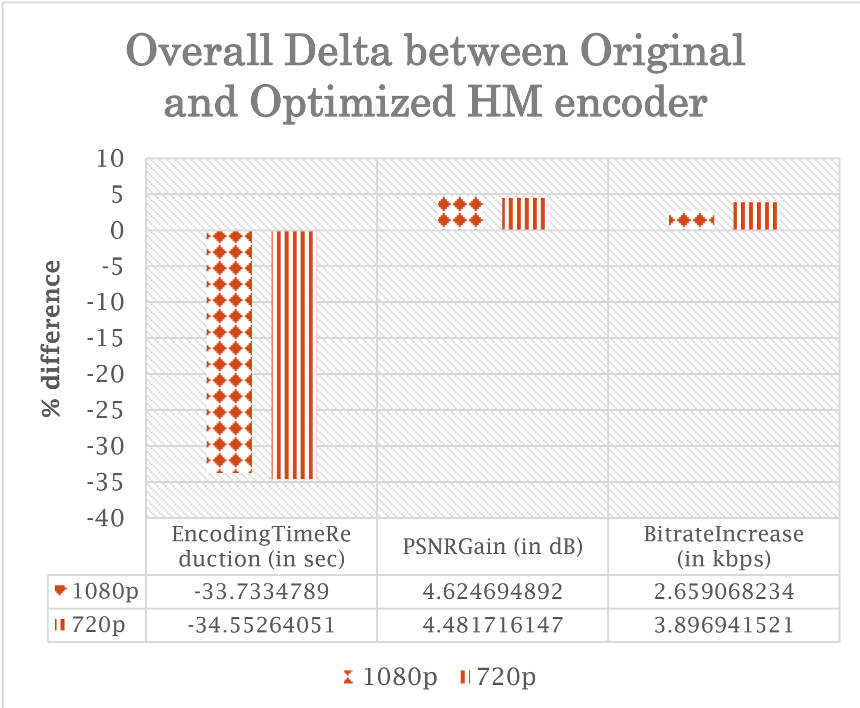
8.1 Conclusions

Through thorough analysis with the most powerful tool, Intel® vTune™ amplifier, hotspots were identified in the HM16.7 encoder. These hotspots are the most time consuming functions/loops in the encoder. The functions are optimized using optimal C++ coding techniques and the loops that do not pose dependencies are parallelized using the OpenMP directives available by default in Windows Visual Studio.

Not every loop is parallelizable. Thorough efforts are needed to understand the functionality of the loop to identify dependencies and the capability of the loop to be made parallel. Overall observation is that the HM code is already vectorized in many regions and hence parallel programming on top of vectorization may lead to degradation in performance in many cases. Thus the results of this thesis can be summarized as below:

- Overall ~24.7 to 42.3% savings in encoding time.
- Overall ~3.5 to 7% gain in PSNR.
- Overall ~1.6 to 4% increase in bitrate.

Though this research has been carried out on a specific configuration (4 core architecture), it can be used on any hardware universally. This implementation works on servers and Personal Computers. Parallelization in this thesis has been done at the frame level.



	EncodingTimeReduction (in sec)	PSNRGain (in dB)	BitrateIncrease (in kbps)
1080p	-33.7334789	4.624694892	2.659068234
720p	-34.55264051	4.481716147	3.896941521

Figure 8-1 Summary of Results

8.2 Future Work

OpenMP framework is a very simple yet easy to adapt framework that aids in thread level parallelism. Powerful parallel programming APIs are available which can be used in offloading the serial code to the GPU. Careful efforts need to be invested in investigating the right choice of software and functions in the software chosen to be optimized. If optimized appropriately, huge savings in encoding time can be achieved.

Intel® vTune™ amplifier is a very powerful tool which makes it possible for analysis of different types to be carried at the code level as well as at the hardware level. The analysis that has been made use of in this thesis is Basic Hotspot analysis. There are other options available in the tool, one of which helps us to identify the regions of the code which cause the maximum number of locks and waits and also the number of cache misses that occur. Microprocessor and assembly level optimization of the code base can be achieved by diving deep into this powerful tool.

Appendix A

List of Acronyms

- ABR: Adaptive Bit Rate
- AMVP: Advanced motion vector prediction
- AVC: Advanced Video Coding
- B: Bi-directionally Predicted Frame
- BD-PSNR: Bjontegaard metric calculation
- CABAC: Context Adaptive Binary Arithmetic Coding
- CAVLC: Context Adaptive Variable Length Coding
- CB: Coding Block
- CIF: Common Intermediate Format
- CPU: Central Processing Unit
- CU: Coding Unit
- CTB: Coding Tree Block
- CTU: Coding Tree Unit
- CUDA: Compute Unified Device Architecture
- DCT: Discrete Cosine Transforms
- DST: Discrete Sine Transform
- FPGA: Field Programmable Gate Array
- GPU: Graphics Processing Unit
- HM: HEVC Model
- HEVC: High Efficiency Video Coding
- I: Intra Frame
- IEC: International Electrotechnical Commission
- ISO: International Organization for standardization
- ITU: International Telecommunication Union
- JCT-VC: Joint Collaborative Team on Video Coding
- MC: Motion Compensation

- ME: Motion Estimation
- MPEG: Moving Picture Experts Group
- MV: Motion Vector
- P: Predicted Frame
- QP: Quantization Parameter
- QCIF: Quarter Common Intermediate Format
- PSNR: Peak Signal To Noise Ratio
- PU: Prediction Unit
- RD: Rate Distortion
- SAO: Sample Adaptive Offset
- SAD: Sum of Absolute Differences
- SATD: Sum of Absolute Transformed Differences (SATD)
- SDK: Software Development Kit
- SHVC: Scalable HEVC
- SIMD: Single Instruction Multiple Data
- SSIM: Structural Similarity
- SVC: Scalable Video Coding
- TU: Transform Unit
- URQ: Uniform Reconstruction Quantization
- VCEG: Video Coding Experts Group
- VOD: Video On Demand

Appendix B

Video Sequences Used

Figure B-1 CrowdRun (Sequence with maximum movements – Hard)

Figure B-2 ParkJoy (Sequence with good about of movements – Medium)

Figure B-3 Ducks Take Off (Sequence with very less movement – Easy)

References

[1] Facebook's Video Boom effects on the Internet consumption - http://recode.net/2014/09/19/look-what-facebooks-video-boom-does-to-the-internet/

[2] Broadcasters biggest web traffic – http://www.theverge.com/2014/2/10/5390620/twitch-one-million-broadcasters-biggest-web-traffic

[3] Bandwidth Explosion – http://arstechnica.com/business/2012/05/bandwidth-explosion-as-internet-use-soars-can-bottlenecks-be-averted/

[4] Mobile Bandwidth Requirements – http://www.rapidtvnews.com/2015021337200/ott-streaming-drives-up-mobile-bandwidth-requirements.html#axzz3onNDvVUk

[5] News about Twitch – http://www.theverge.com/2014/2/10/5390620/twitch-one-million-broadcasters-biggest-web-traffic

[6] K.R.Rao, D.N.Kim and J.J.Hwang, "Video coding standards: AVS China, H.264/MPEG-4 PART 10, HEVC, VP6, DIRAC and VC-1", Springer 2014.

[7] Multimedia Processing Laboratory website - http://www.uta.edu/faculty/krrao/dip/

[8] HEVC Demystified: A Primer on the H.265 Video Codec – https://www.elementaltechnologies.com/resources/white-papers/hevc-h265-demystified-primer

[9] B. Bross et al, "High efficiency video coding (HEVC) text specification draft 8", ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCTVC) document JCTVC-J1003, July 2012.

[10] G. J. Sullivan et al, "Overview of the high efficiency video coding (HEVC) Standard," IEEE Transactions on Circuits and Systems for Video Technology, vol 22 , pp.1649-1668, Dec. 2012.

[11] F. Bossen et al, "HEVC complexity and implementation analysis," IEEE Transactions on Circuits and Systems for Video Technology, vol 22 , pp.1685-1696, Dec. 2012.

[12] H. Samet, “The quadtree and related hierarchical data structures,”Comput. Surv, vol. 16 , pp. 187-260, 1984

[13] N. Purnachand, L. N. Alves and A.Navarro, “Fast motion estimation algorithm for HEVC ,” IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin), 2012.

[14] X Cao, C. Lai and Y. He, “Short distance intra coding scheme for HEVC”, IEEE Picture Coding Symposium, 2012.

[15] M. A. F. Rodriguez, “CUDA: Speeding up parallel computing”, International Journal of Computer Science and Security, Nov. 2010.

[16] M. Abdellah, “High performance Fourier volume rendering on graphics processing units”, M.S. Thesis, Systems and Bio-Medical Engineering Department, Cairo, Egypt, 2012.

[17] Information about quad tree structure of HEVC http://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/

[18] Course on CUDA offered by Stanford University- https://code.google.com/p/stanford-cs193g-sp2010/

[19] M.Budagavi and V.Sze, “Design and Implementation of Next Generation Video Coding Systems (H.265/HEVC Tutorial)”, IEEE International Symposium on Circuits and Systems (ISCAS), June 2014, Melbourne, Australia.

[20] M.J.Jakubowski and G.Pastuszak, “Block-based motion estimation algorithms – a survey ,” Opto-Electronic Review , pp 86-102,Volume 21,March2013.

[21] J. R. Ohm et al, "Comparison of the Coding Efficiency of Video Coding Standards — Including High Efficiency Video Coding (HEVC)," IEEE Trans. CVST, Vol.22, pp.1669 – 1684, Dec. 2012.

[22] H. R. Wu and K. R. Rao, “Digital Video Image Quality and Perceptual Coding,” CRC press, 2006.

[23] M.Zhou, V.Sze and M. Budagavi ,"Parallel Tools in HEVC for High---Throughput Processing,“ SPIE Optical Engineering+Applications, Applicatons of Image Processing XXXV, Vol. 8499, pp. 849910 – 1 to 849910 – 13, 2012.

[24] M. Zhou, et al., “ HEVC lossless coding and improvements”, IEEE Trans. CSVT , vol.22 , pp. 1839 – 1843, Dec. 2012.

[25] V. Sze, M. Budagavi and G.J.Sullivan, “High Efficiency Video Coding (HEVC) Algorithms and Architectures”, Springer 2014.

[26] ITU-T SG16 Q6 and ISO/IEC JTC1/SC29/WG11 (2010) Joint call for proposals on video compression technology. ITU-T SG16 Q6 document VCEG-AM91 and ISO/IEC JTC1/SC29/WG11 document N11113, Kyoto, 22 Jan. 2010

[27] Sullivan GJ and J-R Ohm(2010) Recent developments in standardization of High Efficiency Video Coding (HEVC). In: Proc. SPIE. 7798, Applications of Digital Image Processing XXXIII, no. 77980V, Aug. 2010

[28] T. Wiegand et al (2010), Special section on the joint call for proposals on High Efficiency Video Coding (HEVC) standardization. IEEE Trans. CSVT 20(12):1661–1666

[29] Sullivan GJ, T Wiegand(2005), “Video compression - from concepts to the H.264/AVC standard”, Proc IEEE Vol. 93(1):18–31, Jan. 2005.

[30] T Wiegand, GJ Sullivan, B G, A Luthra (2003), “Overview of the H.264/AVC video coding standard”, IEEE Trans. CSVT 13(7):560–576

[31] ITU-T Rec. H.264 and ISO/IEC 14496-10 (2003) Advanced video coding (May 2003 and subsequent editions).

[32] Sjöberg R et al (2012) Overview of HEVC high-level syntax and reference picture management. IEEE Trans Circuits Syst Video Technol 22(12):1858–187

[33] K. Kim, et al, “Block partitioning structure in the HEVC standard,” IEEE Trans. on circuits and systems for video technology, vol. 22, pp.1697-1706, Dec. 2012.

[34] M.Budagavi et al., “Core Transform Design in the High Efficiency Video Coding (HEVC) Standard,” Vol.7, No.6, pp.1029-1041,IEEE JS TSP, Dec. 2013.

[35] D. Marpe, T. Wiegand and G. J. Sullivan, “The H.264/MPEG-4 AVC standard and its applications”, IEEE Communications Magazine, vol. 44, pp. 134- 143, Aug. 2006.

[36] A. Norkin et al, “HEVC Deblocking Filter”, IEEE Trans. CSVT, Vol. 22, No. 12, pp. 1746-1754, Dec. 2012.

[37] I. E. Richardson, “The H.264 Advanced Video Compression Standard”, 2nd Edition, Wiley 2010.

[38] ARM’s Mali ™ - T600 GPUs - http://www.arm.com/files/event/Mali_Partner_Track_4_GPU_Compute_accelerated_HEVC_decoder_on_ARM_Mali-T600_GPUs.pdf

[39] R.C. Gonzalez and R.E.Woods, “Digital Image Processing”, Pearson, Edition 3, 2009

[40] Test Sequences: ftp://ftp.kw.bbc.co.uk/hevc/hm-11.0-anchors/bitstreams/

[41] S.Cho et al, “HEVC Hardware Decoder Implementation for UHD Video Applications”, IEEE ISCASSP 2014.

[42] M.A. Isnardi,” Historical Overview of video compression in consumer electronic devices”, IEEE ICCE, pp. 1-2, Las Vegas, NV, Jan. 2007.

[43] HEVC tutorial by I.E.G. Richardson: http://www.vcodex.com/h265.html

[44] K. Iguchi et al, “HEVC Encoder for Super Hi-Vision”, 2014 IEEE International conference on Consumer Electronics (ICCE), pp. 61-62, Jan. 2014

[45] F. Pescador et al, “A DSP HEVC decoder implememtation based on Open HEVC”, IEEE ICCE , pp. 65-66, Jan. 2014.

[46] G.J. Sullivan et al,” Standardized Extensions of HEVC”, IEEE Journal of Selected topics in Signal Processing, Vol.7, no.6, pp.1001-1016, Dec. 2013.

[47] BJontegaard metric - http://www.mathworks.com/matlabcentral/fileexchange/27798-bjontegaard-metric/all_files

[48] F. Pescador et al, “ Complexity analysis of an HEVC Decoder based on a Digital Signal Processor”, IEEE Trans. on Consumer Electronics, Vol.59, No.2, pp. 391-399, May 2013.

[49] JCT-VC Video Subgroup, “HM9: High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description”, Shanghai, China, Oct. 2012.

[50] F. Bossen et al, “HEVC Complexity and Implementation Analysis”, IEEE Trans. CSVT, Vol.22, no. 12, pp. 1685 - 1696 , Dec. 2012.

[51] M. Jakubowski and G.Pastuszak, “ Block based motion estimation algorithms – a survey”, Opto-Electronics Review , Vol.21, Issue 1, pp86-102, Dec. 2012.

[52] N. Ahmed, R.Natarajan and K.R.Rao, “Discrete CosineTransform”, IEEE Trans. on Computers, Vol.C-23, pp.90-93, Jan. 1974.

[53] S. Kwon, A. Tamhankar and K.R. Rao, ”Overview of H.264 / MPEG-4 Part 10”, J. Visual Communication and Image Representation, vol. 17, pp.186- 216, April 2006.

[54] I. E. Richardson, “The H.264 Advanced Video Compression Standard”, 2nd Edition, Wiley 2010.

[55] D. Marpe, T. Wiegand and G. J. Sullivan, “The H.264/MPEG-4 AVC standard and its applications”, IEEE Communications Magazine, vol. 44, pp. 134- 143, Aug. 2006.

[56] H.264/MPEG-4 AVC Reference Software Manual - http://vc.cs.nthu.edu.tw/home/courses/CS553300/97/project/JM%20Reference%20Software%20Manual%20(JVT-X072).pdf

[57] HM Software Manual- https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/HM-9.2-dev/doc/software-manual.pdf

[58] HEVC decoder for handheld devices implemented by Ace Thought- http://www.acethought.com/index.php/products/hevc-decoder/

[59] Z. Ma and A. Segall, “Low Resolution Decoding For High-Efficiency Video Coding”, IASTED SIP 2011, Dallas, TX, Dec. 2011.

[60] I.E.G. Richardson, “Video Codec Design: Developing Image and Video Compression Systems”, Wiley, 2002.

[61] HM 16.0 (HEVC Software) Download Link- https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/HM-16.0- dev/

[62] JM 18.6 (H.264 Software) Download Link- http://iphome.hhi.de/suehring/tml/download/

[63] Special Issue on emerging research and standards in next generation video coding, IEEE Trans. CSVT, vol. 22, no. 12, pp.1646- 1909, Dec. 2012.

[64] RC. Gonzalez, RE. Woods, SL. Eddins, “Digital Image Processing using MATLAB”, Gatesmark Publishing.

[65] Imagination company website- http://www.imgtec.com/news/release/index.asp?NewsID=780 [66] NGCODEC website-

http://ngcodec.com/news/2014/2/16/vixs-partners- with-ngcodec-to-license-worlds-first-silicon-proven-10-bit-hevc-decoder-for-ultra- hd-4k-60p

[67] Video Coding: HEVC and Beyond- IEEE Journal of Selected Topics In Signal Processing, vol.7, pp. 931-1151, Dec. 2013.

[68] H. Zhang and Z.Ma , “Fast Intra Mode Decision for HEVC”, IEEE Trans. on CSVT, vol. 24 , pp.660-668, April 2014.

[69] T.Wiegand and G.J.Sullivan, “The picture phone is here. Really”, IEEE Spectrum, vol. 48, pp.51-54, Sept. 2011.

[70] NVIDIA CUDA C Programming Guide- http://www.cs.unc.edu/~prins/Classes/633/Readings/CUDA_C_Programming_Guide_4.2.pdf

[71] CUDA H.264 by NVIDIA- http://www.mainconcept.com/products/sdks/gpu-acceleration-sdk/cuda-h264avc.html#product_page-5

[72] White Paper Real-Time CPU Based H.265/HEVC Encoding Solution with Intel® Platform Technology: https://software.intel.com/sites/default/files/white_paper_real-time_HEVC_encodingSolution_IA_v1.0.pdf

[73] AES Department of Electrical and Computer Engineering- http://www.aes.tu-berlin.de/

[74] Website to access JCTVC Documents: http://www.itu.int/en/ITU- T/studygroups/2013-2016/16/Pages/video/jctvc.aspx

[75] J. Dubashi, “Complexity reduction of H.265 motion estimation using CUDA (Compute Unified Device Architecture)”, Thesis, EE Department, University of Texas at Arlington, 2014 – http://www.uta.edu/faculty/krrao/dip/Courses/EE5359/index_tem.html

[77] M.Budagavi, “Design and Implementation of Next Generation Video Coding Systems HEVC/H.265 Tutorial”, Seminar presented in EE Department, UTA, 21 Nov 2014. (http://iscas2014.org/)

[78] OpenCL tutorial link - https://developer.nvidia.com/opencl

[79] Special Issue on Screen Content Video Coding and Applications, IEEE Journal on Emerging and Selected Topics in Circuits and Systems(JETCAS), Final manuscripts due on 22nd July 2016.

[80] K.R.Rao, “A tutorial on HEVC”.

[81] Tutorial on finding hotspots using Intel® vTune™ Amplifier – https://software.intel.com/sites/default/files/hotspots_amplxe_lin.pdf

[82] W. Hamidouche et al, “4K Real-Time and Parallel Software Video Decoder for Multi-layer HEVC Extensions”, IEEE Trans. On CSVT early access.

[83] G. Correa et al, “Pareto-Based Method for High Efficiency Video Coding with Limited Encoding Time”, IEEE Trans. On CSVT early draft.

[84] W. Zhao et al, “Hierarchical Structure-Based Fast Mode Decision for H.265/HEVC”, IEEE Trans. On CSVT, Vol.25, No.10, Oct. 2015.

[85] Video sequences download link - https://media.xiph.org/video/derf/

[86] Matlab code for MSE and PSNR implementation - http://emanuelecolucci.com/2011/04/image-and-video-quality-assessment-part-one-mse-psnr/

[87] Lecture Notes on Parallel Programming by Dr.Roger S. Walker, Professor of Computer Science and Engineering at The University of Texas at Arlington – http://ranger.uta.edu/~walker/

Biographical Information

Vasavee Vijayaraghavan was born in Chennai, Tamil Nadu, India. After completing her schooling at school in Chennai in 2007, she went on to obtain her B.E in Electrical and Electronics Engineering from MNM Jain Engineering College from 2007 - 2011. From 2011 to 2013, she worked as a Systems Engineer with Infosys Technologies, Chennai.

She joined University of Texas at Arlington to pursue her M.S in Electrical Engineering in 2013. This was around the time she joined the Multimedia Processing Lab. She is presently working as a Summer/Fall Graduate Engineer intern at Intel Corporation, Oregon and subsequently will join the same team after she graduates.